L1 MOS

COBOL
Program Preparation and Execution

olivetti

PUBLICATION ISSUED BY:

Ing. C. Olivetti & C., S.p.A.
Direzione Documentazione
77, Via Jervis - 10015 IVREA (Italy)

PREFACE

This manual provides the information required to prepare and execute application programs written in COBOL on L1 MOS systems.

SUMMARY

The manual is divided into chapters which describe the stages of program preparation, compilation, linking and execution. Further chapters detail the compatibility between ICE COBOL and COBOL and the services accessible from COBOL. The two indexes list the compilation error texts and execution time error texts.

REFERENCES

Read first ...

Introduction to MOS - Code 4002130 G (Vol. 2)

MOS SHELL Commands Reference Manual - Code 4002770 Q (Vol. 3)

For further information read ...

COBOL Language Reference Manual - Code 4004290 H (Vol. 6d)

MOS - EDITOR - Reference Manual - Code 4002440 P (Vol. 6f)

MOS - MCL MOS Command Language - User Guide - Code 4002220 H (Vol. 3)

MOS - Program Development Tools - Reference Manual
Code 4002790 S (Vol. 6f)

Transaction Handler Configuration Guide - Code 4003150 E (Vol. 7a)

VISA (S6000 - Compatible) Programmer Guide
Code 4004590 P (Vol. 6g)

VISA - Form Management Package - Programmer Guide
Code 4004390 B (Vol. 6g)

CSS Line Manager Interface - Programmer Guide
Code 4000800 B (Vol. 6g)

PGU - Graphics Management Package - Programmer Guide
Code 4004650 D (Vol. 6g)

PASCAL+ . Program Preparation and Execution - Code 4002480 T (Vol. 9a)

MTS - Programmer Guide - Code 4003410 Q (Vol. 6h)

MTS - Configuration Guide - Code 4003450 L (Vol. 7a)

MOS - Message Book - Code 4002260 D (Vol. 3)

MOS Programmer Guide - Code 4002570 L (Vol. 6g)

MOS ONE User Guide - Code 4008540 B

BEAM - User Guide - Code 4003180 R (Vol. 6h)

CAT - Applicative Interface - Programmer Guide
Code 4022060 G (Vol. 6g)

FIRST EDITION: March 1984 - Release 3.0

UPDATES: September 1984 - Release 4.0
January 1985 - Release 4.1
May 1985 - Release 5.0
December 1985 - Release 5.1

REPRINTED: August 1986

(Including updates from 1 to 4)

UPDATE: February 1987 - Release 5.2

CONTENTS

2. PROGRAM PREPARATION

SELECTING THE WORKING DIRECTORY

The user should set the required working directory, using the Shell SETWDIR command.

```
MCL: SETWDIR wdirname
```

SOURCE PREPARATION

The source code should be prepared using the MOS System Editor in a manner to conform with the COBOL language syntax. The source code must be in card format. The editor is accessed from SHELL using the command:

```
MCL: EDITOR filename
```

The editor opens or creates a file with the specified name in the current working directory.
Editing is done using the functions that are available, noting that card format permits columns columns 73 to 80 to be blank or contain the card sequence number.
For a complete description of the editor refer to the " MOS - Editor - Reference Manual".

SOURCE COMPILATION

The source code must be compiled using the COBOL compiler. The COBOL compiler and library routines are supplied in a directory named "/IPL/DPC". Access to this directory must be enabled before compilation by setting the path to it as shown:

```
MCL: SET %PATH := '/IPL/DPC'
```

This action may be included in the user's terminal initialization procedure.

COMPILER INVOCATION

The compiler is accessed from Shell specifying the options required in any order on the command line as follows:

```
MCL: COBOL S=filename OPTION OPTION ...
```

The compiler validates the options and any file names specified in the command string. Errors in the specification of any of these result in an error message on the screen i.e.

UNRECOGNIZABLE OPTION 'XX' or

CANNOT OPEN 'filename' or

FILENAME 'extralongfilename' TOO LONG

The command line must be re-entered specifying valid options.

Concurrent compiler invocations under the same working directory are not supported.

System errors during compilation

If a compilation is aborted due to a system error (such as a file system error) a message is displayed on the screen. The message shows the type of error, the system class and code.

Example:

```
I-O ERROR        CLASS = xx    CODE = yy
```

where:
CLASS is a decimal value indicating the system reply class.
CODE is a decimal value indicating the system reply code.

The most common are:

CLASS =1 CODE = 5

Meaning: the number of files on the volume on which the compilation was being performed is insufficient.

CLASS = 1 CODE = 6

Meaning: the disk space on the volume on which the compilation was being performed is insufficient. For possible values and meanings of class and code, see L1 MOS Message Book manual.

Compilation in Batch mode or in a Procedure

The compiler may be executed in batch mode using the Shell BM command or in a Shell MCL procedure.

The compiler sets a value in the system register STATUS to indicate the outcome of a compilation.

Compilation Completion Codes

0 = Compilation correct or containing warning messages.
1 = Compilation contains mild errors and possible warning messages the assumptions made by the compiler should be checked.
2 = Compilation contains severe and/or fatal errors; no object code is generated.

4. If the L= or O= option is not specified then the last string of the source file name not considering suffixes may be at most 10 characters in length to allow for the '.L' and '.O' suffixes.

Compiler Limits

The COBOL compiler has the following limits:

- the maximum number of compilable lines is 32727
- the maximum space reserved for the Symbol Table is 512 pages. Each page occupies 512 bytes.

The number of lines compiled and the number of pages occupied by the Symbol Table are shown in the listing file, in the following messages:

NUMBER OF COMPILED LINES: nnnnnn

NUMBER OF SYMBOL TABLE PAGE(S) USED: xxxxx OUT OF 512

where:
nnnnn indicates the number of lines compiled
xxxxx indicates the number of pages used by the Symbol Table.

Note: the number of pages in the Symbol Table cannot be defined by the user. The message is given simply for information.

COMPILATION ERROR MESSAGES

The compilation errors are classified in five categories according to their severity:

- **INFO** Used primarily to indicate the point at which error recovery finished and syntax analysis resumed, thus providing the user with a guide as to the amount of source, if any, that was bypassed due to the source error.
- **WARNING** Used to indicate minor source code errors which have little impact on the compiler's ability to process the source. The user should make sure that the compiler's assumption was the one intended.
- **MILD** Used to indicate an error where the compiler's assumption is probably correct but requires verification by the user. (eg. unequal length REDEFINES areas where the compiler assumes the larger).
- **SEVERE** Used to indicate an error that causes the object code file not to be created.
- **FATAL** Used to indicate that either a compiler limitation has been reached, a compiler internal error has been encountered, or an extremely malformed program has been submitted for compilation.

The compile time error messages and warnings, if not suppressed, are printed with the compiler statistics at the end of all the other requested compiler listings.

A complete list of compilation error messages is given in Appendix A "Compilation Error Messages".

COMPILATION OUTPUT

The compiler creates two output files:

- A listing file
- An object code file.

The listing file name is specified in the L= option, if not specified a file is created, in the present working directory with a name obtained by the addition of the suffix '.L' to the source file name, after the removal of any existing suffix.
The listing file contains the listings of all the list outputs requested, error messages and compiler statistics.

The object file name is specified in the O= option, if not specified a file is created, in the present working directory, with a name obtained by the addition of the suffix '.O' to the source file name, after the removal of any existing suffix.
The object file is not created if severe errors are detected during compilation.

EXAMPLES OF OUTPUT LISTINGS

1. SOURCE LISTING

```
NSPL COBOL COMPILER    0006.002          COBOL SOURCE LISTING

        L-----CA---B-----------------------------------------------------R--------
      1         IDENTIFICATION DIVISION.
      2         PROGRAM-ID.
      3             CB45190.
      4         AUTHOR.
      5             CONTROLLO QUALITA' NUOVA LINEA SISTEMI
      6             OLIVETTI  IVREA.
      7         SECURITY.
      8           -- NONE.


     73         01  TEST-RESULTS.
     74             02 FILLER                      PICTURE X.
     75             02 FEATURE                     PICTURE X(20).
     76             02 FILLER                      PICTURE X.
     77             02 P-OR-F                      PICTURE X(5).
     78             02 FILLER                      PICTURE X.
     79             02  PAR-NAME.
     80               03 FILLER PICTURE X(12).
     81               03  PARDOT-X PICTURE X.
     82               03 DOTVALUE PICTURE 99.
     83             02 FILLER                      PICTURE X.
     84             02 COMPUTED-A                  PICTURE X(20).
     85             02 COMPUTED-N REDEFINES COMPUTED-A PICTURE -9(9).9(9).
     86             02 COMPUTED-0V18 REDEFINES COMPUTED-A  PICTURE -.9(18).
     87             02 COMPUTED-4V14 REDEFINES COMPUTED-A  PICTURE -9(4).9(14).
     88             02 COMPUTED-14V4 REDEFINES COMPUTED-A  PICTURE -9(14).9(4).
     89             02       CM-18V0 REDEFINES COMPUTED-A.
     90                 03 COMPUTED-18V0                  PICTURE -9(18).
     91                 03 FILLER                         PICTURE X.

    172         01  TABGLO IS  EXTERNAL GLOBAL.
    173             02  ELEMGLO  OCCURS 3 TIMES INDEXED BY INDXGLO.
    174                 04  PRIMO     PIC X(9).
    175                 04  SECONDO   PIC X(9)
    176                 04  TERZO     PIC XX.
    177         *


    184         PROCEDURE    DIVISION.
    185         CCVS1 SECTION.
    186         OPEN-FILES.
    187             OPEN     OUTPUT PRINT-FILE.
    188             MOVE "CB45190 " TO TEST-ID.
    189             MOVE     TEST-ID TO ID-AGAIN.
    190             MOVE    SPACE TO TEST-RESULTS.
    191             PERFORM  HEAD-ROUTINE THRU COLUMN-NAMES-ROUTINE.
    192             GO TO CCVS1-EXIT.
    193         CLOSE-FILES.
    194             PERFORM END-ROUTINE THRU END-ROUTINE-3.
    195             CLOSE    PRINT-FILE.
    196         TERMINATE-CCVS.
```

OBJECTS GENERATED BY THE COMPILER

This section describes the relationship between objects generated by the COBOL compiler and objects handled by OLINK. Subsequently the various sections generated by program functions are listed and briefly described, and in conclusion the user is given a number of practical hints in order to avoid overflows from OLINK tables.

The objects generated by the compiler are:

- symbols
- sections
- references

The definitions of these objects are shown in the Program Development Tools Reference Manual.

SYMBOLS GENERATED BY THE COMPILER

During the compilation phase of a COBOL program, the compiler generates a symbol, the name of the program (PROGRAM-ID).

SECTIONS GENERATED BY THE COMPILER

During compilation of a COBOL program, the compiler generates a number of sections. Of these, some are always generated (default sections), and others are generated depending on the contents of the program.

The sections are divided in:

- Sections identified by a fixed name.
- Sections identified by the name of the program (PROGRAM-ID) preceded by a 3-character prefix indicating the section type and followed by a suffix indicating the progressive number of the section. The prefix may be absent.

The maximum length of a section is 64K bytes. If a section is not big enough to contain all the data or the coding with the same characteristics, the compiler automatically generates a new section.

Sections Generated by Default

The sections generated by default, identified by a fixed name and/or prefix, are:

- a section named __cobol_ovlnum

- a section named _cob_dummysec
- a section named _cobol_progtab
- a section named _cobol_ptbllen
- a section with the prefix ME_
- a section with the prefix ST_
- a section without prefix
- a section with the prefix CD_
- a section with the prefix SK_

Sections Depending on the Source

The following list shows the sections generated as a result of the presence in the program of certain functions. The list shows the name of the function and the type and number of sections generated by it.

FILE SECTION	generates a section identified by the prefix FI_.
SAME RECORD AREA	generates a section identified by the prefix SA_.
File EXTERNAL	every EXTERNAL file generates a section identified by the name of the file itself, in lower-case letters.
WORKING-STORAGE SECTION	generates one or more sections identified by the prefix US_.
INDEXES	if there are tables defined with the clause INDEXED BY, a section is generated identified by the prefix IN_.
EXTERNAL variables	every variable at 01 level defined EXTERNAL generates a section identified by the name of the variable itself, in lower-case.
LINKAGE SECTION	generates a section identified by the prefix LI_.
SCREEN SECTION	generates two sections, the first identified by the prefix SC_, the second identified by the prefix CD_.
REPORT SECTION	generates a section identified by the prefix RE_.
CONSTANTS	generate a section identified by the prefix CN_.

PROCEDURE DIVISION	there is no connection between the SECTION defined in the source and the sections generated by the compiler. One or more sections identified by the prefix CD_ are generated.
CALL DATA-NAME	generates, regardless of whether the instruction implies dynamically loading a link unit, the following sections: - a _cob_srhblks section - a _cob_srhtmp section - a _cob_srhname section - a section with the prefix CN_.
EDITED MOVE	if the MOVE instruction has a receiving field defined with an edited PICTURE, a section identified by a name without prefix is generated.
MERGE	generates a section identified by the prefix CD_.
EMULATED VISA	generates a section identified by the name VISA-TBL.
NESTED PROGRAMS	every nested program generates two sections identified by the prefix CD_.
THS	if the program is compiled with the option THS two sections are generated, the first with the prefix CH_, the second with the prefix SH_.

REFERENCES GENERATED BY THE COMPILER

During compilation of a COBOL program, the compiler generates a number of run time procedure references. The number of references depends on the program instructions.

It is impossible to give the exact number of references generated by the compiler, as it depends entirely on the source program. It is only possible to give the minimum and maximum number of references generated by the compiler.

If, for example, a program is made up of just an empty PROCEDURE DIVISION, the compiler generates three external symbol references.

If a program uses all the functions of the language, the compiler generates about 300 external references, and possibly another 60 references if the Emulated VISA is used.

When counting the references it is also necessary to bear in mind that:

- for every literal CALL instruction calling external programs, in COBOL or in other languages, the compiler generates a reference.

- for every call to a service contained in the EXTINTF.LIB library, the compiler generates a reference.

RUN TIME CONTRIBUTION

In counting the objects generated by the compiler it is necessary to consider also the objects that are linked to the program when carrying out the link to the run time COBOL. The number of objects linked to the COBOL program are:

- 600 symbols

- 60 sections

- 220 references

The run time contribution is not under the direct control of the user and the contribution of the called EXTINTF.LIB modules must be added to it.

OPTIMIZATION OF THE OLINK TABLES

As stated at the start of this section, it may happen that in effecting a COBOL application link made up of several modules, an overflow of the OLINK tables occurs. To avoid this problem it is necessary to:

1. reduce the number of sections and references generated by the compiler

2. reduce the number of programs linked together by the linker.

To reduce the number of sections and references it is necessary to: specialize the programs making up an application and reduce the number of variables declared EXTERNAL, grouping them at a single 01 level.

To reduce the number of programs linked together it is necessary to structure the COBOL applications in link units, and effect an OLINK command on each one (see the section LINK UNIT STRUCTURES).

- MAP - A file containing all the listings requested and the link error diagnostic messages.

- o01 - to o0n, Any overlay load files specified.

If the linker is unable to create executable code because of errors encountered or is unable to create a program directory, or aborts it creates or overwrites a file named LINK.MAP in the current working directory. This file contains all the warning and error messages concerning the current link.

The linker is described in the "MOS - Program Development Tools - Reference Manual".

SOME OLINK COMMANDS

Among the OLINK commands are the following:

- REGION - ensures that the specified region becomes the current region.

- USE - specifies a list of segment numbers that the linker may use for the current region. The USE command, if present, is always associated to the REGION command. The REGION and USE commands are used when it is necessary to indicate the number of unusable segments, because already occupied.

- RETAIN - specifies the symbols that must be held on the EPL file. The RETAIN command must be linked to the EPL option. If the RETAIN command is followed by "*" it identifies any name.

- RETAINSEC - specifies the sections that must be held on the EPL file. The RETAINSEC command must be linked to the EPL option.

- DELETESEC - specifies the sections with the OVERLAY attribute that are to be deleted from the EPL file.

Note: The definitions and the limitations of all the commands and the objects (segments, sections etc.) handled by OLINK are contained in the manual Program Development Tools - Reference Manual. The same manual contains the "name" terms and the "global symbols" which are equivalent to the COBOL terms "user-defined terms" and "user-defined terms also recognized outside the module".

OVERLAY STRUCTURES

The structuring of any COBOL program in overlays is accomplished by compiling the objects of the programs separately and using the linker to create the desired overlay structure.

The structure of a program in overlays permits the main program to call a program in an overlay. The overlay program may only call another program in the same overlay, a program in an overlay of another region, or return to the main program.

Take for example the following structure:

A root program MAIN1.0 with four called programs CPA.0, CPB.0, CPC.0, CPD.0, each compiled separately.
In which MAIN1.0 calls CPA.0, CPB.0 and CPC.0, and in turn CPA.0 and CPB.0 both call CPD.0.

The following two sets of linker instructions would not produce link time errors, but the first would produce an executable program and the second would not.

Example 1:

```
root
     MAIN1.0
     CPA.0

region  RANYNAME

   overlay  OANYNAME
                    CPB.0
                    CPD.0

   overlay  OVANY2
                    CPC.0
```

These instructions would produce an executable program as the calls involved return to the root segment to permit the run time to load the second overlay.

Example 2:

```
root
     MAIN1.0

region  RANYNAME

   overlay  OANYNAME
                    CPA.O
                    CPB.O

   overlay  OVANY2
                    CPC.O
                    CPD.O
```

These instructions would not produce an executable program as the calls involved attempt to invoke one overlay from another in the same region.

The overlay structure is not permitted if the root segment is written in PASCAL+ and calls COBOL subroutines.

For a complete description of overlay specification refer to the MOS - Program Development Tools - Reference Manual .

THE STRUCTURE OF LINK UNIT

A COBOL application may be structured to have a primary module and one or more secondary modules. The primary module is referred to as root link unit, the secondary modules which are dynamically loadable are referred to as link unit. The link units may be loaded into one or more memory regions. For a description of a memory region refer to "The generalized linker (OLINK)" chapter in the manual MOS - Program Development Tools - Reference Manual.
A link unit may contain one or more a object programs, but only one may be the entry-point of the link unit. The entry-point object program permits access to the link unit.

The root link unit may have a structure containing overlays in which case the root module and the overlay modules must be generated in one link process. Link units may not have any overlay mechanism.

As opposed to a program with an overlay structure, each link unit is generated by one link process. The time taken to modify an application program is less if the modification affects a link unit and not a object program in root link unit. Modification of an object program not in the root link unit requires only recompilation and relinking of the containing link unit.

LINK UNIT ACCESS

The calling mechanism between link units is effected using the CALL statement.

The CALL literal

The CALL statement with a literal argument may be used when the calling and called object programs are in the same link unit, or when the called object program is in the root link unit. The names of all the object programs in a root link unit are communicated to the link units by the link process.

The CALL data-name

The CALL statement with a data-name argument permits dynamic loading in memory of a link unit and the passing of control to it.

The CALL statement searches for the specified data-name in the current link unit; if found control is passed to the named object program; if not found the CALL data-name interprets the contents of data-name as the path name of a file in another link unit. The search for the file is as follows:

- If a complete path name starting with the file system root is specified, the search starts from the root directory.

- If the path name specified in data-name is a partial path name, the search starts from the program directory associated with the root link unit.

- If a partial path name is specified and no file is found under the program directory, the first name in the path is considered to be the name of a library directory connected through Shell and the search is started from that directory.

For a Standard conforming COBOL program containing the CALL data-name statement, the string contained in data-name must be the same as the PROGRAM-ID of the entry-point object program of the associated link unit. In which case the link unit must be contained in the program directory of the root link unit.

If the search process is successful the link unit associated with the file is loaded into its assigned memory region and control is passed to the entry-point object program of the link unit, referred to as the called object program. The other object programs forming part of a link unit may only be called from within the link unit.

If the search is unsuccessful the caller is returned an error code which may be tested by the application program.

PARAMETER PASSING

The CALL data-name with dynamic loading permits the specification of a list of parameters by use of the USING phrase. The parameters are passed to the called program by reference.

The CALL data-name places no limit on the maximum number of parameters which may be passed.

SHARING OF DATA AND/OR FILES

A link unit can share data and/or files with the root link unit. The shared data and/or files must be declared using the EXTERNAL mechanism. Every data item and/or file defined with the EXTERNAL attribute in a link unit must also be declared in an object program in the root link unit. The linking of the link unit, which shares data and/or files with the root link unit, must be effected when the names and the allocation of the data and/or the shared files are available; it must therefore be effected after that of the root link unit. To ensure correct handling at the OLINK level, of data and/or files declared EXTERNAL, the RETAINSEC command must be specified for each item declared as EXTERNAL. The use of a common prefix or suffix for all items declared as EXTERNAL permits the use of a single RETAINSEC command for the linker by declaring the common prefix or suffix together with the character '*' (eg. RETAINSEC *.EXT).

The data items and/or files defined with the EXTERNAL attribute are available to all link units which define the same data-name as EXTERNAL.

EXCEPTION HANDLING

Error conditions can occur during the processing of a CALL statement, such as: a link unit with the specified name does not exist, or the memory region specified for the link unit is in use.

If the ON OVERFLOW/EXCEPTION phrase is specified in the CALL data-name statement for certain errors control passed to the imperative statement.

The special register MESSAGE contains a code indicating the type of exception.

If the ON OVERFLOW/EXCEPTION phrase is not specified and an error occurs, the application program is aborted.

Three exception cases result in an abort of the program whether or not the ON OVERFLOW/EXCEPTION phrase was specified:

1. A call to a link unit which has not been released by a previous call.

2. The number of parameters in the calling and called object programs differ.

3. Unexpected system error.

DEALLOCATION OF LINK UNIT

In order to load a link unit into a memory region already occupied by a link unit the region must be freed by use of the CANCEL statement.

The CANCEL literal

The CANCEL statement with a literal argument may be used when the cancelling and cancelled object programs are in the same link unit.

The compiler option CNCL must be specified at compile time for the link unit to be cancelled.

The CANCEL data-name

The CANCEL statement with a data-name argument searches the current link unit
for the string specified in data-name. If the search is successful the object program is cancelled. For the object program to be cancelled properly, the compiler option CNCL must have been specified at compile time. If the search is unsuccessful the string specified in data-name is taken as the path name to a file in another link unit. The search process is the same as that specified for the CALL data name.

If a file with the given path name is found, all the object programs in the link unit are cancelled and the link unit is de-allocated. No error results if the link unit to be de-allocated is not found or has not been loaded.

In order to de-allocate a link unit, the compiler option CNCL need not be specified for any of the object programs in the link unit.

For a complete description of the statements CALL and CANCEL refer to the COBOL - Reference Manual.

THE COBOL LINKER COMMANDS FILE

The following section describes the contents of the command files that must be supplied to OLINK for carrying out the COBOL application link. The command files described here are distributed with run time COBOL.

The command files distributed with run time COBOL are as follows:

- RTLINK
- RTLINK.ROOTLU
- RTLINK.INTLU
- RTLINK.OVLU

The RTLINK Command File

The linker command file, /IPL/DPC/COB_RTS/LIB/RTLINK supplied with run time COBOL contains:

```
REGION USERPACKAGE USE 40 41 42 43 44 45 46 47
ROOT
     /IPL/DPC/COB_RTS/LIB/RTINTID
     /IPL/DPC/COB_RTS/LIB/NUCL.UI.OBJ
     /IPL/DPC/COB_RTS/LIB/RTIO.UI.OBJ
     /IPL/DPC/COB_RTS/LIB/SCRH.UI.OBJ
     /IPL/DPC/COB_RTS/LIB/SORT.UI.OBJ
     /IPL/DPC/COB_RTS/LIB/RPWR.UI.OBJ
     /IPL/DPC/COB_RTS/LIB/VISA.UI.OBJ
     /IPL/DPC/COB_RTS/LIB/RTS.EPL
     /IPL/DPC/COB_RTS/LIB/COBOL.LIB
     /IPL/DPC/COB_RTS/LIB/EXTINTF.LIB
     /IPL/DPC/COMMON/LIB/SYS.LIB
```

Where the specified interface libraries contain:

- RTINTID - control mask of the run time COBOL configuration.
- NUCL.UI.OBJ - contains support for a minimum language capacity for processing data inside the structure of a COBOL program.
- RTIO.UI.OBJ - contains support for all the instructions of the "SEQUENTIAL MODULE", the "RELATIVE MODULE" and the "INDEXED MODULE".
- SCRH.UI.OBJ - contains support for the "SCREEN HANDLING MODULE".

- SORT.UI.OBJ - contains support for the "SORT-MERGE MODULE".

- RPWR.UI.OBJ - contains support for the "REPORT WRITE MODULE".

- VISA.UI.OBJ - contains support for the VISA derived from S6000 (VISA EMULATED).

- RTS.EPL - permits reserving the segments used by run time COBOL.

- COBOL.LIB - contains the run time COBOL routines used for initialization, debugging, and stop run.

- EXTINTF.LIB - contains the external interface libraries that carry out connections with BEAM, VISA, MTS etc.

- SYS.LIB - contains the system interface libraries used by run time COBOL.

The meaning and the contents of the specified segments in the REGION USE command are described in the manual MOS - Programmer Guide.

The RTLINK file is used for carrying out linking of COBOL applications non-structured in link units. The information contained in the file is necessary for carrying out the link at run time and for the allocation of the segments in memory.

The RTLINK.ROOTLU File Commands

The RTLINK.ROOTLU command file contains the information necessary for effecting the link of the root link unit present in a COBOL application structured in link units. This file contains the information necessary for effecting the link at run time and for the generation of the EPL file which is used in the command supplied to OLINK for effecting the link of the link units allocated in regions different from the current ones. The contents of the file is as follows:

```
REGION USERPACKAGE USE 40 41 42 43 44 45 46 47
ROOT
     /IPL/DPC/COB_RTS/LIB/RTINTID
     /IPL/DPC/COB_RTS/LIB/NUCL.UI.OBJ
     /IPL/DPC/COB_RTS/LIB/RTIO.UI.OBJ
     /IPL/DPC/COB_RTS/LIB/SCRH.UI.OBJ
     /IPL/DPC/COB_RTS/LIB/SORT.UI.OBJ
     /IPL/DPC/COB_RTS/LIB/RPWR.UI.OBJ
     /IPL/DPC/COB_RTS/LIB/VISA.UI.OBJ
     /IPL/DPC/COB_RTS/LIB/RTS.EPL
     /IPL/DPC/COB_RTS/LIB/COBOL.LIB
     /IPL/DPC/COB_RTS/LIB/EXTINTF.LIB
     /IPL/DPC/COMMON/LIB/SYS.LIB
     RETAIN *
```

At the link phase the RTLINK.ROOTLU command file generates the EPL file containing information on the segments occupied by the root link unit, and information on the allocation of the compilation units forming the root link unit. The RTLINK.ROOTLU command file may be used instead of the RTLINK file, but not vice-versa.

The RTLINK.INTLU Command File

The RTLINK.INTLU command file is used for effecting the link of the intermediate link unit. By "intermediate link unit" is meant a link unit different from the root link unit that is found at a level hierarchically above other link units and containing compilation units that can be called by link units hierarchically lower. The contents of the file is as follows:

```
REGION USERPACKAGE USE 40 41 42 43 44 45 46 47
     /IPL/DPC/COB_RTS/LIB/RTS.EPL
     /IPL/DPC/COB_RTS/LIB/EXTINTF.LIB
     /IPL/DPC/COMMON/LIB/SYS.LIB
     RETAIN *
```

At the link phase, the RTLINK.INTLU command file generates the EPL file containing information on the segments occupied by the intermediate link unit, and information on the allocation of the compilation units contained in the intermediate link unit.

RTLINK.OVLU Command File

The RTLINK.OVLU command file is used for effecting the link of link units of a level hierarchically below all the others, or of link units that do not contain compilation units that can be called. The contents of the file is as follows:

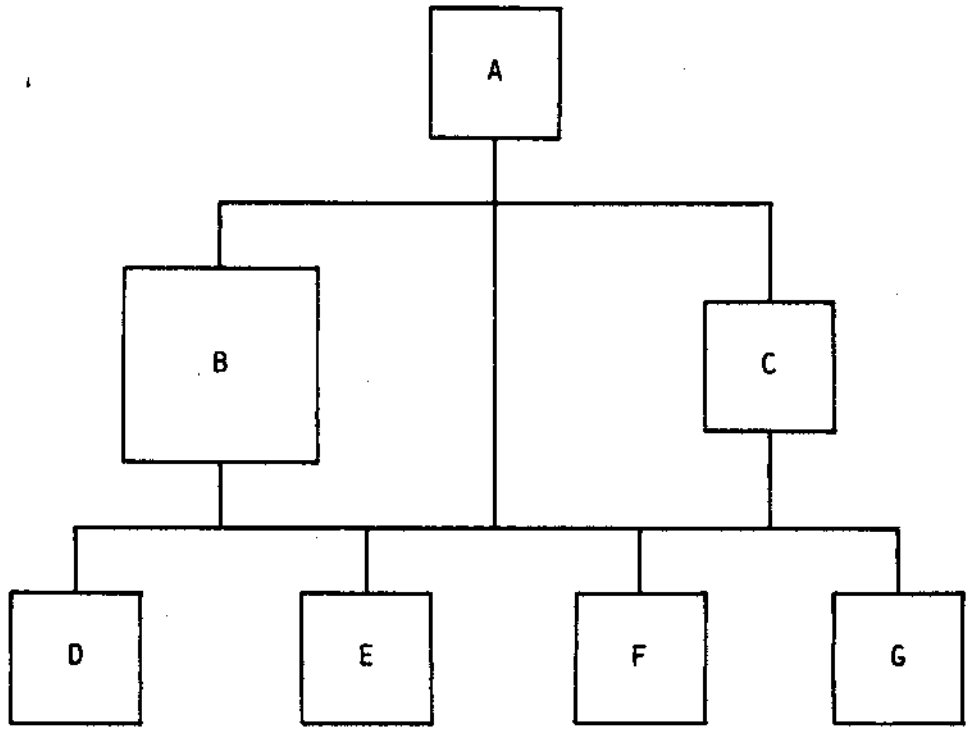
```
REGION USERPACKAGE USE 40 41 42 43 44 45 46 47
     /IPL/DPC/COB_RTS/LIB/RTS.EPL
     /IPL/DPC/COB_RTS/LIB/EXTINTF.LIB
     /IPL/DPC/COMMON/LIB/SYS.LIB
```

At the link phase, the RTLINK.OVLU generates the EPL file containing information on the segments occupied by the link unit, if the EPL option has been specified.

EXAMPLES OF LINK UNIT STRUCTURED COBOL APPLICATIONS

The following examples show two COBOL applications structured to link unit, with the relative commands necessary for OLINK.

Example 1



The application is made up of a root link unit A, two link units B and C allocated in region 1, and four link units D E F and G allocated in region 2. Each single link unit may be made up of more than one compilation unit. This link unit structured application is not organized hierarchically; that is, a link unit of a region may call each link unit of any region. For example C may call D, A may call E, F may call C etc.

It is in no way possible to call a link unit from another link unit of the same region.

The compilation units contained in the root link unit may be called by each link unit of Region 1 and region 2. The compilation units contained in the link units of region 1 and 2 cannot be called.

Note that the set of compilation units of each X link unit is identified with X.0.

OLINK command for root link unit A.

```
OLINK options EPL ... A.0 RTLINK.ROOTLU
```

OLINK generates A/EPL, which contains information on the segments used and on the compilation units defined in A.

OLINK command for link unit B.

```
OLINK options EPL ...B.0 A/EPL RTLINK.OVLU
```

OLINK generates B/EPL, which contains information on the segments used by the link unit.

OLINK command for link unit C:

```
OLINK options EPL ... C.0 A/EPL RTLINK.OVLU
```

OLINK generates C/EPL, which contains information on the segments used by the link unit.

OLINK command for link unit D:

```
OLINK options ... D.0 A/EPL x/EPL RTLINK.OVLU
```

File x/EPL is chosen from among those generated by B and C, using the one corresponding to the link unit that occupies most segments. In the example given, file B/EPL must be used. OLINK does not generate the EPL file.

OLINK command for link unit E:

```
OLINK options ... E.0 A/EPL x/EPL RTLINK.OVLU
```

File x/EPL is chosen from among those generated by B and C, using the one corresponding to the link unit that occupies most segments. In the example shown, the B/EPL file must be used.

OLINK command for link unit F:

```
OLINK options ... F.0 A/EPL x/EPL RTLINK.OVLU
```

File x/EPL is chosen from among those generated by B and C, using the one corresponding to the link unit that occupies most segments. In the example shown, file B/EPL must be used. OLINK does not generate the EPL file.

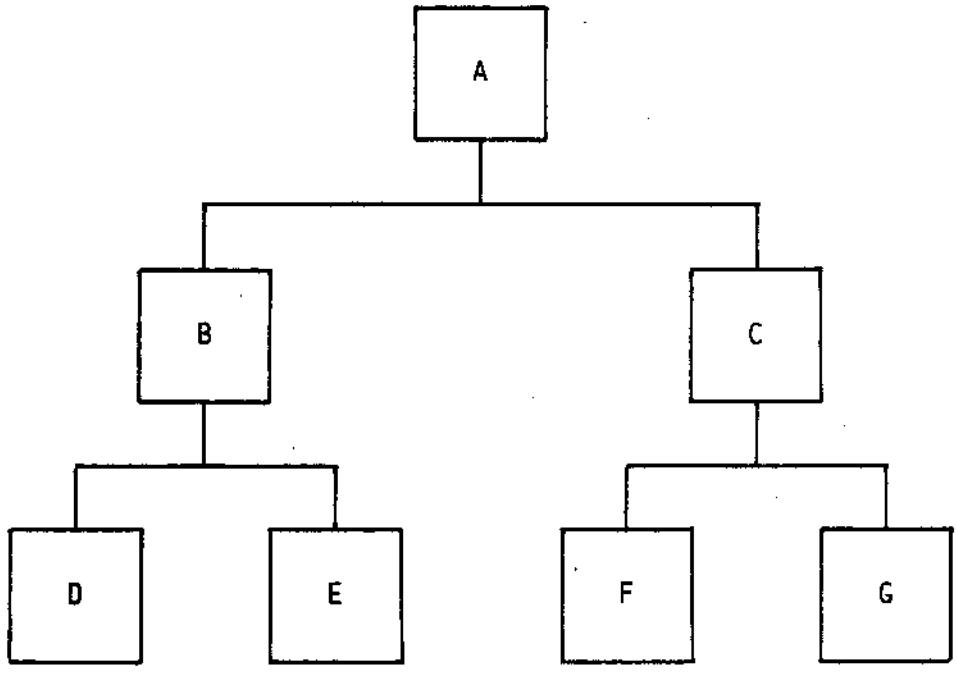
OLINK command for link unit G:

```
OLINK options ... G.O A/EPL x/EPL RTLINK.OVLU
```

File x/EPL is chosen from among those generated by B and C, using the one corresponding to the link unit that occupies most segments. In the example shown, file B/EPL must be used. OLINK does not generate the EPL file.

A modification to the root link unit always necessitates linking all link units. A modification to a link unit of region 1 necessitates linking the link units of region 2 only if the modification involves changing the x/EPL file. A modification to a link unit of region 2 necessitates linking only the link unit modified.

Example 2



The application is made up of a root link unit A, an intermediate link unit B and a link unit C allocated in region 1, and four link units D, E, F and G allocated in region 2. Each single link unit may be made up of more than one compilation unit. This application is organized hierarchically; that is, each link unit of any level may only call certain link units of lower levels. For example, link unit A may only call link unit B or C, intermediate link unit B may only call link units D or E, and link unit C may only call link units F or G.

In no way is it possible to call a link unit from another link unit of the same region.

The compilation units contained in the root link unit may be called by each link unit of region 1. The compilation units contained in

intermediate link unit B may be called by link units D or E. The compilation units contained in link unit C are not called. The compilation units contained in the link units of region 2 cannot be called.

OLINK commands

Note that the set of compilation units of each X link unit is identified with X.0.

OLINK command for root link unit A:

```
OLINK options EPL ... A.0 RTLINK.ROOTLU
```

OLINK generates A/EPL, which contains information on the segments used and the compilation units contained in A.

OLINK command for intermediate link unit B:

```
OLINK options EPL ... B.0 A/EPL RTLINK.INTLU
```

OLINK generates B/EPL, which contains information on the segments used and on the compilation units contained in B.

OLINK command for link unit C:

```
OLINK options EPL ... C.0 A/EPL RTLINK.OVLU
```

OLINK generates C/EPL, which contains information on the segments used by the link unit.

OLINK command for link unit D:

```
OLINK options ... D.0 A/EPL B/EPL RTLINK.OVLU
```

OLINK does not generate the EPL file.

OLINK command for link unit E:

```
OLINK options ... E.0 A/EPL B/EPL RTLINK.OVLU
```

OLINK does not generate the EPL file.

OLINK command for link unit F:

```
OLINK options ... F.0 A/EPL C/EPL RTLINK.OVLU
```

OLINK does not generate the EPL file.

OLINK command for link unit G:

```
OLINK options ... G.O A/EPL C/EPL RTLINK.OVLU
```

OLINK does not generate the EPL file.

A modification to the root link unit necessitates linking all link units. A modification to intermediate link unit B necessitates linking link units D and E. A modification to link unit C necessitates linking link units F and G only if the segments occupied by C are increased. A modification to a link unit of region 2 necessitates linking only by the link unit modified.

EXAMPLES OF USE OF CALL WITH DYNAMIC LOADING OF LINK UNITS

The COBOL application described here is made up of an overlay structured root link unit, and of five link units. The root link unit is made up of one main and four overlays: BETA, CHROMA, DELTA and GAMMA.

The sequence show in the diagram below represents the first cycle of calls and returns starting from the CHROMA overlay. This cycle is repeated for all the other overlays and for the main.

In order to fully understand the diagram the following should be borne in mind:

1. The symbol  indicates the main.

2. The symbol  indicates the overlay.

3. The symbol  indicates the link unit.

The continuous lines indicate calls to link units or overlays.

The dotted lines indicate returns to link units or overlays.

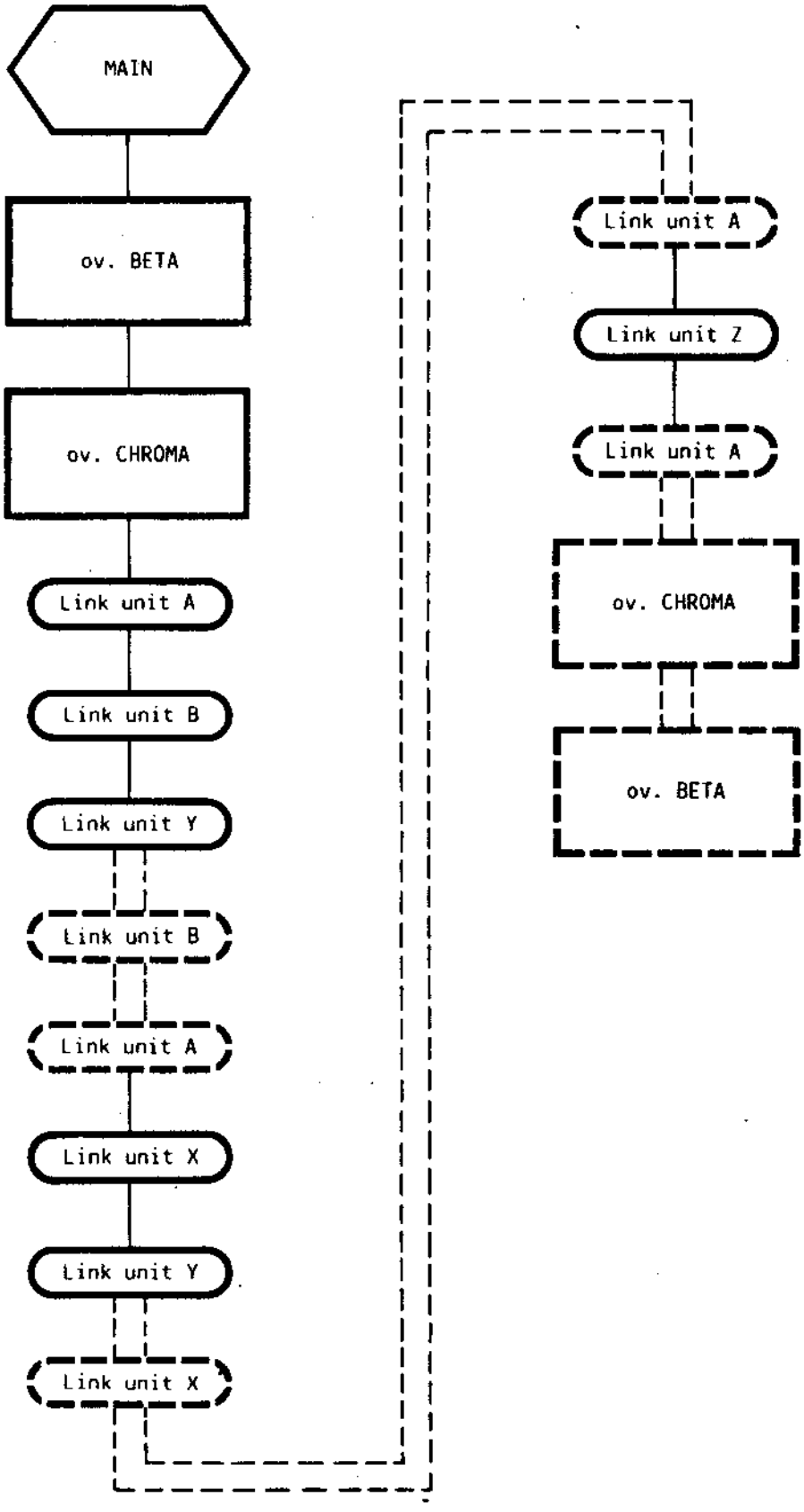


Fig. 2-1 Rappresentation of the First Cycle of Calls between Overlays and Link Unit in the Following Example

Below are shown the source listings of the COBOL application.

Principal program (MAIN)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CB4444A.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  D-CB 444A1   PIC X(7) VALUE "CB444A1".
       PROCEDURE DIVISION.
       AA.
           DISPLAY "I root link unit".
           DISPLAY "start program".
       BB.
           CALL "CB4444B".
       XX.
           CALL "CB4444X".
       ZZ.
           CALL "CB4444Z".
       AA1.
           CALL  D-CB444A1 ON OVERFLOW
                 DISPLAY "message(1) = " MESSAGE(1)
                 DISPLAY "message(2) = " MESSAGE(2).
      EE.
           DISPLAY "end of program".
           DISPLAY "F       root link unit".
           STOP RUN.
```

Overlay GAMMA

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CB4444Z.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  D-CB444A1   PIC X(7)  VALUE "CB444A1".
       PROCEDURE DIVISION.
       AA.
           DISPLAY "root link unit region THEMA overlay GAMMA".
           DISPLAY "start subprogram Z".
           CALL D-CB444A1 ON OVERFLOW
                DISPLAY "message(1) = " MESSAGE(1)
                DISPLAY "message(2) = " MESSAGE(2).
           DISPLAY "end subprogram Z".
       BB.
           EXIT PROGRAM.
```

Overlay BETA

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CB4444X.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  D-CB444A1   PIC  X(7)  VALUE "CB444A1".
PROCEDURE DIVISION.
AA.
    DISPLAY "root link unit region PRISMA overlay DELTA".
    DISPLAY "start subprogram X".
    CALL "CB4444Y".
    CALL D-CB444A1 ON OVERFLOW
         DISPLAY "message(1) = "  MESSAGE(1)
         DISPLAY "message(2) = "  MESSAGE(2).
    DISPLAY "end subprogram X".
BB.
    EXIT PROGRAM.
```

Overlay DELTA

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CB4444B.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  D-CB444A1   PIC  X(7)  VALUE "CB444A1".
PROCEDURE DIVISION.
AA.
    DISPLAY "root link unit region PRISMA overlay BETA".
    DISPLAY "start subprogram B".
    CALL "CB4444Y".
    CALL D-CB444A1 ON OVERFLOW
         DISPLAY "message(1) = "  MESSAGE(1)
         DISPLAY "message(2) = "  MESSAGE(2).
    DISPLAY "end subprogram B".
BB.
    EXIT PROGRAM.
```

Overlay CHROMA

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CB4444Y.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  D-CB444A1   PIC  X(7)  VALUE "CB444A1".
PROCEDURE DIVISION.
AA.
    DISPLAY "root link unit region THEMA overlay CHROMA".
    DISPLAY "start subprogram Y".
    CALL D-CB444A1 ON OVERFLOW
         DISPLAY "message(1) = "  MESSAGE(1)
```

```
            DISPLAY "message(2) = "  MESSAGE(2).
        DISPLAY "end subprogram Y".
    BB.
        EXIT PROGRAM.
```

Link Unit A

```
    IDENTIFICATION DIVISION.
    PROGRAM-ID. CB444A1.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01  D-CB444B1   PIC  X(7)  VALUE "CB444B1".
    01  D-CB444X1   PIC  X(7)  VALUE "CB444X1".
    01  D-CB444Z1   PIC  X(7)  VALUE "CB444Z1".
    PROCEDURE DIVISION.
    AA.
        CANCEL D-CB444X1.
        CANCEL D-CB444Z1.
        DISPLAY "start link unit A".
        CALL D-CB444B1 ON OVERFLOW
             DISPLAY "message(1) = "  MESSAGE(1)
             DISPLAY "message(2) = "  MESSAGE(2).
    BB.
        CANCEL D-CB444B1.
        CALL D-CB444X1 ON OVERFLOW
             DISPLAY "message(1) = "  MESSAGE(1)
             DISPLAY "message(2) = "  MESSAGE(2).
    CC.
        CALL D-CB444Z1 ON OVERFLOW
             DISPLAY "message(1) = "  MESSAGE(1)
             DISPLAY "message(2) = "  MESSAGE(2).
    DD.
             DISPLAY "end link unit A".
             EXIT PROGRAM.
```

Link Unit B

```
    IDENTIFICATION DIVISION.
    PROGRAM-ID. CB444B1.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01  D-CB444Y1   PIC  X(7)  VALUE "CB444Y1".
    PROCEDURE DIVISION.
    AA.
        DISPLAY "start link unit B".
        CALL D-CB444Y1 ON OVERFLOW
             DISPLAY "message(1) = "  MESSAGE(1)
             DISPLAY "message(2) = "  MESSAGE(2).
        DISPLAY "end link unit B".
    BB.
        EXIT PROGRAM.
```

Link Unit X

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID. CB444X1.
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        01  D-CB444Y1   PIC  X(7)  VALUE "CB444Y1".
        PROCEDURE DIVISION.
        AA.
            DISPLAY "start link unit X".
            CALL D-CB444Y1 ON OVERFLOW
                 DISPLAY "message(1) = "  MESSAGE(1)
                 DISPLAY "message(2) = "  MESSAGE(2).
            DISPLAY "end link unit X".
            CANCEL D-CB444Y1.
        BB.
            EXIT PROGRAM.
```

Link unit Y

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID. CB444Y1.
        AA.
            DISPLAY "start link unit Y".
            DISPLAY "end link unit Y".
        BB.
            EXIT PROGRAM.
```

Link Unit Z

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID. CB444Z1.
        AA.
            DISPLAY "start link unit Z".
            DISPLAY "end link unit Z".
        BB.
            EXIT PROGRAM.
```

OLINK Commands

The following are the commands required by OLINK to perform the link of the link of the above example application program.

```
OLINK PROGRAM CB4444A.E OPTIONS EPL MEMORY

ROOT
       CB4444A.O

REGION PRISMA
       OVERLAY BETA
                          CB4444B.O
       OVERLAY DELTA
                          CB4444X.O

REGION THEMA
       OVERLAY CHROMA
                          CB4444Y.O
       OVERLAY GAMMA
                          CB4444Z.O

       /IPL/DPC/COB_RTS/LIB/RTLINK.ROOTLU;

OLINK PROGRAM CB444A1.E OPTIONS MEMORY EPL
              CB444A1.O CB4444A.E/EPL
              /IPL/DPC/COB_RTS/LIB/RTLINK.INTLU;

OLINK PROGRAM CB444B1.E OPTIONS MEMORY EPL
              CB444B1.O CB4444A.E/EPL CB444A1.E/EPL
              /IPL/DPC/COB_RTS/LIB/RTLINK.INTLU;

SHOW 'CB444X1' ;

OLINK PROGRAM CB444X1.E OPTIONS MEMORY
              CB444X1.O CB4444A.E/EPL CB444A1.E/EPL
              /IPL/DPC/COB)RTS/LIB/RTLINK.INTLU;

OLINK PROGRAM CB444Y1.E OPTIONS MEMORY
              CB444Y1.O CB4444A.E/EPL CB444A1.E/EPL CB444B1.E/EPL
              /IPL/DPC/COB_RTS/LIB/RTLINK.OVLU;

OLINK PROGRAM CB444Z1.E OPTIONS MEMORY
              CB444Z1.O CB4444A.E/EPL CB444A1.E/EPL CB444B1.E/EPL
              /IPL/DPCCOB_RTS/LIB/RTLINK.OVLU;
```

EXAMPLE OF OLINK COMMANDS FOR LINK UNIT CONTAINING EXTERNAL DATA AND/OR FILES

The following is an example of OLINK commands to correctly perform the link of a COBOL application, structured with link unit containing data and/or file declared as EXTERNAL.

```
OLINK PROGRAM CB44111.E OPTIONS EPL MEMORY

      RETAINSEC '*'
      DELETESEC '_cob*'

      CB44111.O  /IPL/DPC/COB_RTS/LIB/RTLINK.ROOTLU ;

OLINK PROGRAM CB44111C1.E OPTIONS EPL MEMORY
      CB44111C1.O CB44111.E/EPL
      /IPL/DPC/COB_RTS/LIB/RTLINK.INTLU ;

OLINK PROGRAM CB44111C2.E OPTIONS EPL MEMORY
      CB44111C2.O CB44111.E/EPL CB44111C1.E/EPL ;
      /IPL/DPC/COB_RTS/LIB/RTLINK.INTLU ;

OLINK PROGRAM CB44111C3.E OPTIONS MEMORY
      CB44111C3.O CB44111.E/EPL CB44111C1.E/EPL CB44111C2.E/EPL
      /IPL/DPC/COB_RTS/LIB/RTLINK.OVLU ;
```

3. PROGRAM EXECUTION

PROGRAM INVOCATION

The working directory must be set to the name of the load module directory containing the executable code.

Program execution is started by entering the program directory name.

```
MCL: dirname
```

Execution begins with the initialization routine of the COBOL run time then control passes to the application program.

If the COBOL run time does not exist or the memory size is insufficient a message of the form:

```
COBOL RUN TIME DOES NOT EXIST
or
SYSTEM ERROR LOADING COBOL RUN TIME
```

will be displayed.

TERMINAL INTERACTION

The ACCEPT statement places the cursor on the screen and awaits an input terminated by an enter key. The program continues execution with the next executable statement.

The STOP 'literal' statement results in the display of the literal on the screen and a wait for acknowledgement. Pressing the enter key causes the program to continue execution with the next executable statement.

The STOP RUN statement results in termination of the program and a return to the Shell or other father environment.

Run time error messages are displayed on the current cursor line except when the screen was defined in the SCREEN SECTION, in which case the error messages are displayed on the penultimate physical line of the screen.

SCREEN SIZE COMPATIBILITY

The compiler checks the dimensions of screens specified in the SCREEN SECTION, the maximum size permitted is 24 lines by 80 columns.

A COBOL program may be executed on any type of terminal.
It is only possible to dynamically alter screen formats in COBOL using the TFORM facilities from VISA.
The COBOL run time routines ensure congruence between the declared format and the physical screen at all times.

RUN TIME ERRORS

Run time errors are classified as:

- Warning.
- Run time errors with status code 1-0.
- Run time errors without status code 1-0.
- Errors detected by the operating system.

Occurrence of a warning causes the display of a message pointing out anomalies which do not cause the abortion of the COBOL application. The text of the warning is self-explanatory.

Run time errors with status code 1-0 may be handled by the COBOL application specifying in the declarative part of the Procedure Division (DECLARATIVE) a procedure that recognizes the 1-0 status code concerned and permits the COBOL application to continue. In the event of not specifying the procedure that recognizes the 1-0 status code, occurrence of run time errors with 1-0 status code causes the display of an error message, the closure, by the run time, of all files previously opened, abortion of the COBOL application and return to the environment from which execution was requested.

Run time errors without 1-0 status code may be handled by the COBOL application if the error concerns the CALL instruction, associating to this the sentence ON OVERFLOW or ON EXCEPTION. Occurrence of every other type of run time error without 1-0 status code causes the display of an error message, the closure, by run time, of all files previously opened, abortion of the COBOL application and return to the environment from which execution was requested.

During execution of a COBOL application, a system error is possible; this causes closure by the run time of all files previously opened, abortion of the COBOL application and return to the environment from which execution was requested; in addition, the following message is displayed:

SYSTEM REPLY UNEXPECTED "system error"

where "system error" is a string describing the system error concerned.

Some errors that occur at operating system level display a message of type:

```
*** PASCAL RUN TIME ERROR  system error ***
```

where "system error" is a string describing the system error that has occurred, whose meaning is described in the manual System Software Maintenance, User Guide.

Errors detected by the run time routine are indicated by means of displaying messages of type:

```
COBOL RUN TIME ERROR
 instruction * FILE IS file name or text in error
```

A complete list is found in Appendix B "Run Time Error Messages".

EXECUTION UNDER SYMBOLIC DEBUG

Starting a program under control of debug is accomplished by entering

```
MCL: DEB progname
```

If the program requires parameters or switches, the directive PRG must be used as follows:

```
MCL: DEB PRG=progname params
```

DEBUG starts by outputting its current version number and waiting for a command to be entered.

The Debug commands include breakpoints, traces, memory contents display, memory and register contents alteration by symbolic name or absolute address.

For a complete explanation of Symbolic Debug and its use refer to the MOS - Program Development Tools, Reference Manual";

If the program was compiled and/or linked without specifying the Internal Symbol Dictionary option, DEBUG will commence and output the message

```
DEB version x.xx  ISD NOT FOUND
```

In this case debugging may be performed but only using commands referring to absolute addresses.

Execution in Batch Mode or in a Procedure

Any program may be executed in batch mode from Shell using the BM command. For example:

```
MCL: BM progname
```

In batch mode the output destined for the screen will be lost unless it is redirected to a file. The form of the BM command is:

```
MCL: BM >outfile progname
```

If the COBOL program requests the use of the system spooler facilities the connection of the appropriate files must be made within the scope of the BM command, this is accomplished by submitting a file of commands. For example:

```
MCL: BM afile
```

where afile contains the following:

```
CONN SYSPRNT1 SPOOL2
progname
```

Programs may be executed in Shell MCL procedures, the COBOL run time system sets a value to indicate the outcome of the execution in the system register STATUS.

Run time completion codes

0 = Normal program termination no run time errors detected.

1 = Warning errors issued to the program, ie. out of range values in the VALUE OF clause for SPLIT-STRATEGY, TIME-OUT, ALLOCATION-UNIT, or FILE-SIZE.

2 = Errors detected, the program terminated abnormally. An appropriate message is sent to the standard output device with the I/O status value if the error was caused by an I/O operation.

PROGRAM OUTPUT

OUTPUT TO PRINTER

In the SELECT statement for a file if ASSIGN TO SYSPRINTx is specified, where x refers to a printer number, connection is made to the physical device upon execution of the OPEN statement. If the device is busy (exclusive use by another user) or not ready the associated USE procedure will be invoked after a wait of two minutes. The I-O status item will be set to 90. If no USE procedure has been defined the user program will be aborted.

When a printer is available to the program it is opened in exclusive mode.

The use of the WITH NO LOCK phrase in the OPEN statement causes the printer to be opened in shared mode, in this case other output may be made on the same printer concurrently.

OUTPUT FILE SPOOLING

A file may be spooled for output to printer by ASSIGNing it to a filename in the SELECT statement and before execution of the program using the SHELL CONNECT command to connect the filename to SPOOLx, where x is the number of the desired printer.
Each printer on a system will normally be allocated a spool printer number. For example:
If the desired printer was allocated spool number 2, output to it is enabled by CONNecting the filename specified in the ASSIGN clause to SPOOL2.

```
MCL: CONN outfile SPOOL2
```

Attachment of the file to the spool queue is activated at the time of closing the file itself (specified in the connection/assignment phase).

FILE MAPPING

The COBOL file organizations are mapped onto MOS file system types as follows:

COBOL Organization	MOS file system type	
SEQUENTIAL TEXT	BYTE STREAM	
SEQUENTIAL	POSITIONAL	(no delete)
RELATIVE	POSITIONAL	(delete)
INDEXED	KEYED	(keys internal)

File System Constraints

The number of files OPEN at one time and the number of files or records LOCKED at one time depends upon the system configuration.

SORTING and MERGING FILES

The Sorting and Merging of files may be specified taking into account the following rules:

- They may only be specified for files with SEQUENTIAL organization.
- A maximum of 16 input files may be specified.
- The maximum number of sort keys is 60 and the size of the key block is limited to 256 bytes.

PROGRAM TUNING CONSIDERATIONS

In addition to the rules that apply to any programming language for improving program execution time and reducing object code size, here are some hints relating to COBOL.

- For improving execution time:

  . The arithmetic operations USAGE is COMP, COMP-4, BINARY or WORD-BINARY are preferred to other types.

  . The mixing of data types in statements is to be avoided.

  . The SYNCHRONIZED clause should be used for data item alignment.

  . Initialization of items by VALUE should be preferred to that by MOVE.

- For reducing object code size:

  . The use of the Internal Symbol Dictionary for debugging purposes should be removed when possible.

- For improving program writing:

  . Use of the GLOBAL and EXTERNAL phrases to define areas shared by several programs; this is preferable to passing parameters, when there are many sub-programs.

    The CALL statement with dynamic loading should be used for programs which need to be frequently modified but do not involve many object programs. Execution time of a structured-to-link COBOL application program is virtually the same as the execution time of an overlay-structured COBOL application.

- To avoid loss of data:

  . When working in asynchronous mode it is advisable to effect, as soon as possible, a CLOSE operation in the event of a power failure, or inadvertent machine switching-off.

4. COBOL INTERFACES WITH OTHER LANGUAGES AND SERVICES

SERVICES

Programs written in COBOL may be linked with programs written in other languages or other services. The access to the services is effected using a CALL statement in the program and linking the object code with the interface library containing the references to the services.

NOTE: When services or environments are interfaced from COBOL the function keys assume the significance assigned by the particular service or environment. The manual describing the service or environment should describe the significance of the function keys.

COBOL AND MTS SERVICES

The structure of the data items in the USING phrase of the CALL statement must be defined according to the requested services. The special register MESSAGE is updated with a code indicating the outcome of the execution of the CALL statement. For a description of each service see the MTS Programmer Guide manual.

The COBOL application programs and the user written services must be compiled for a Transaction Handler Environment, by specifying the 'TH' option for the compilation.

The MTS standard services are obtained by linking the program with the interfaces: MTS_ui.obj, MTS.obj, MTS_BATCH.obj, CHAIN_DML.obj and MWS_ui.obj. These are included in the EXTINTF.LIB library which is linked with the user's COBOL program using the linker commands supplied in the /IPL/DPC/COB_RTS//LIB/RTLINK file.

The user written services should be compiled and linked individually each forming a program load module.

Prior to execution of the application program, access to the user written services referred to should be established. During the configuration of the environment for MTS the component and characteristics of the user written services should be entered in a table using the utility 'MTSCONF'. This table is used by the runtime routines to resolve the calls to the services requested. For further information, refer to the MTS Configuration Guide manual.

COBOL AND VISA S6000 COMPATIBLE SERVICES

The data items in the USING phrase of the CALL statement must be defined according to the service requested. Examples and descriptions of the services are given in the VISA S6000 - Compatiple User Guide manual.

The file DDPARM must exist in the current working directory and must contain the complete pathname of the forms library omitting the first '/' character.

There are no required options for the compilation.

There are no required options for the linker. The VISA S6000 services are included with the COBOL run time routines if it is a COBOL with VISA S6000 system, and are linked with the user's COBOL program using the linker commands supplied in the /IPL/DPC/COB_RTS/LIB/RTLINK/b file.

COBOL AND VISA SERVICES

The data items in the USING phrase of the CALL statement must be defined according to the service requested. Examples and descriptions of services are given in the VISA - Form Management Package - Programmer Guide manual.

There are no required options for the compilation.

The VISA services are obtained by linking the program with the interface VISA.obj. This is included in the EXTINTF.LIB library which is linked with the user's COBOL program using the linker commands supplied in the /IPL/DPC/COB_RTS/LIB/RTLINK file.

The VISA MULTIPROCESS services are obtained by linking the program with the interface VISAP.obj. This is included in the EXTINTF.LIB library which is linked with the user's COBOL program using the linker commands supplied in the /IPL/DPC/COB_RTS/LIB/RTLINK file.

COBOL AND LINE MONITOR SERVICES

The data items in the USING phrase of the CALL statement must be defined according to the service requested. Examples and descriptions are given in the CSS - Line Manager Interface - Programmer Guide manual.

There are no required options for the compilation.

The LINE MONITOR services are obtained by linking the program with the interface LM_ui.obj. This is included in the EXTINTF.LIB library which is linked with the user's COBOL program using the linker commands supplied in the /IPL/DPC/COB_RTS/LIB/RTLINK file.

COBOL AND BEAM SERVICES

The structure of the data items in the USING phrase of the CALL statement must be defined according to the requested services. Examples and a complete description are contained in the manual BEAM - Programmer Guide.

No compilation options are required.

The BEAM services are obtained by linking the program with the interfaces bconfl.obj and blog.obj. These are included in the EXTINTF.LIB library which can be linked with the user's programs using the linker commands supplied in the /IPL/DPC/COB_RTS/LIB/RTLINK file.

COBOL AND THE PGU - GRAPHIC MANAGEMENT PACKAGES

The structure of the data items in the USING phrase of the CALL statement must be defined according to the service requested. Examples and a description of each service is given in the manual PGU - Graphic Management Package - Programmer Guide.

No compilation options are required.

The PGU graphic services can be obtained by linking the program with the object code for the graphic section and the files to the character type table; these are called FONT0.....FONT8 and are under the directory /IPL/DPL/GRAPHICS. The interface CO_PGU_ui.obj that is in the EXTINTF.LIB library is linked to the COBOL user programs in the same operation, using the linker commands in the /IPL/DPC/COB_RTS/LIB/RTLINK file.

DRAWING A GRID ON THE SCREEN

A grid can be drawn on the screen in COBOL, using special characters.

The structure of the data items in the USING phrase of the CALL statement must be defined according to the service requested. Examples and a description of these services is given in the manual MOS - Programmer Guide.

COBOL AND RECOVERY SERVICES

The recovery services are obtained by using the Commit Manager which provides the possibility of maintaining coherency in the Data Base by following a sequence of operations. Examples and descriptions of services are given in the manual MOS - Programmer Guide.

The standard services of the Commit Manager are obtained by linking the program with the interface CM_ui.obj which is included in the EXTINTF.LIB library and is linked with the COBOL user program using the linker commands supplied in the /IPL/DPC/COB_RTS/LIB/RTLINK file.

The recovery procedures are requested by using the CALL statement with the service name. The data items in the USING phrase must be defined according to the same service.

COBOL AND SIGNATURE VERIFICATION

The signature verification procedures permits the entry and display of a signature.

The structure of the data items in the USING phrase of the CALL statement must be defined according to the requested services. Examples and a complete description are contained in the manual MOS - Programmer Guide.

No compilation options are required.

The signature verification services are obtained by linking the program with the interfaces SCANNER.OBJ and DECOMP.OBJ. These are included in the EXTINTF.LIB library which is linked with the user's COBOL program using the linker commands supplied in the file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL AND MESSAGE SWITCHING SERVICES

Message Switching services permit the exchange of messages between active applications resident on different nodes of a distributed configuration.

The data of the USING phrase of the CALL statement must be defined according to the service requested; for examples, and a complete description, see the manual MOS - Programmer Guide.

Compilation options are not required.

The Message Switching services are obtained by linking the program with the interface MSW_ui.obj, resident in the EXTINTF.LIB library, which may be linked with COBOL programs using the linker commands present in the file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL AND ERROR LOGGING SERVICES

The error logging services are obtained by the LMS (Local Management System) whose modules must be defined in the system configuration.

The structure of the data items in the USING phrase of the CALL statement must be defined according to the services requested. For examples and a complete description see the manual MOS - Programmer Guide.

No compilation options are required.

The standard LMS services are obtained by linking the program with the interface lms_ui.obj that is in the EXTINTF.LIB library, that is linked to the COBOL user programs by using the linker commands supplied in the /IPL/DPC/COB_RTS/LIB/RTLINK file.

COBOL AND CONNECTION TO NETWORK ONE

To use the services that are obtained by connection to the ONE network the user must access or the module SLAM or the module SNAM using the CALL statement, followed by the required primitive; the data in the USING phrase must be consistent with the service. For examples and a description of the service see the manual MOS - ONE User Guide.

No compilation options are required.

The standard SLAM services are obtained by linking the program with the linker commands supplied in the /IPL/DPC/COB_RTS/LIB/RTLINK file.

The standard SNAM services are obtained by linking the program with the linker commands supplied in the /IPL/DPC/COB_RTS/LIB/RTLINK file.

COBOL AND CAT WORKSTATION

Connection to a workstation used in banking applications CAT (Customer Activated Terminal) can to made in a COBOL application program. For a description of the latter and of the programming modes, see the manual CAT - Applicative Interface - Programmer Guide.

No compilation options are required.

The standard CAT services are obtained by linking the program with the interfaces eru_ui.obj, INTAPP.obj and INTAMB. These are included in the EXTINTF.LIB library which is linked with user's COBOL program using the linker commands supplied in the /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL AND LOGOFF SERVICES

The LOGOFF function suspends a connection, on a switched line, between a remote workstation and the system, so saving telephone costs.

The data items in the USING phrase of the CALL statement must be defined according to the requested services. Example and a complete description are contained in the manual MOS - Programmer Guide.

No compilation options are required.

The LOGOFF services are obtained by linking the program with the interface WSR.lib. These are included in the EXTINTF.LIB library, that is linked with the user's COBOL program using the linker commands supplied in the file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL AND RS232/CL

Permits the handling of peripherals connected to the RS232/CL interface.

The structure of the data items in the USING phrase of the CALL statement must be defined according to the requested services. Example and a complete description are contained in the manual RS232/CL Interface - Programmer Guide.

No compilation options are required.

The RS232/CL services are obtained by linking the program with the interfaces TUB_ui.obj and rs_UI.obj. These are included in the EXTINTF.LIB which is linked with the user's COBOL program using the linker commands supplied in the file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL AND GSP - GRAPHIC SUBROUTINE PACKAGE

The GSP package performs graphics output and is oriented towards commercial applications.

The structure of the data items in the USING phrase of the CALL statement must be defined according to the requested services. Examples and a complete description are contained in the manual GSP - GRAPHING SUBROUTINE PACKAGE - Application Interface Programmer Guide.

No compilation are required.

The GSP services are obtained by linking the program with the interface RTGSP_ui.obj. These are included in the EXTINTF.LIB library which is linked with the user's COBOL program using the linker commands supplied in the file /IPL/DPC/COB_RTS/LIB/RTLINK.

```
           02 FLAG1 PIC 9 COMP.
           02 INT2 PIC S9(4) COMP-4 SYNC.
           02 ENTYPE1 PIC 9(2) COMP.
```

2. The Pascal+ variant record:

```
type variants =
     (long, twoInts, fourChars, fourFlags);

type T_four_Characters = record
     char1, char2, char3, char4: char;
     end;

type T_four_Flags = record
     flag1, flag2, flag3, flag4: boolean;
     end;

type examp3 = record
     commonPart: integer;
     case tag_field : variants of
     long:
          (long1: longinteger);
     twoInts:
          (int1, int2: integer);
     fourChars:
          (four_Characters: T_four_Characters);
     fourFlags:
          (four_Flags: T_four_Flags);
     end;
```

can be mapped into the following COBOL record description:

```
01 EXAMP2-GROUP.
   03 COMMONPART PIC S9(4) COMP-4.
   03 TAG-FIELD PIC 9(2) COMP.
   03 LONG1 PIC S9(9) COMP-4 SYNC.
   03 TWOINTS REDEFINES LONG1.
      06 INT1 PIC S9(4) COMP-4 SYNC.
      06 INT2 PIC S9(4) COMP-4 SYNC.
   03 FOUR-CHARACTERS REDEFINES LONG1.
      06 CHAR1 PIC X.
      06 CHAR2 PIC X.
      06 CHAR3 PIC X.
      06 CHAR4 PIC X.
   03 FOUR-FLAGS REDEFINES LONG1.
      06 FLAG1 PIC 9 COMP.
      06 FLAG2 PIC 9 COMP.
      06 FLAG3 PIC 9 COMP.
      06 FLAG4 PIC 9 COMP.
```

while the following, apparently equivalent:

```
type variants =
     (long, twoInts, fourChars, fourFlags);

type examp2 = record
```

```
commonPart: integer;
case tag_field : variants of
long:
     (long1: longinteger);
twoInts:
     (int1, int2: integer);
fourChars:
     (char1, char2, char3, char4: char);
fourFlags:
     (flag1, flag2, flag3, flag4: boolean);
end;
```

cannot be mapped into the same COBOL record description.

GUIDE RULES FOR PASCAL+ PROCEDURES

These guide rules should be followed when writing Pascal+ procedures to be called by COBOL programs.

- Write only procedures, as functions cannot be called.
- Do not use "flexible" or "conformant" arrays as parameters.
- Do not use "value" parameters.
- Use parameter types listed in the tables above.
- Do not use packed records as parameters. If they can be mapped once, this does not imply that the mapping is correct forever.
- Avoid as much as possible the use of variant records as parameters, since mapping them sometimes can be troublesome.

Graphic-type or Line Monitor Packages

Access to certain of the PASCAL+ system routines is directly available as they are in the library COB_RTS/LIB/SYS.LIB which is included in the linker commands supplied as DPC/IPL/RTLINK.

PASCAL+ TO COBOL

It is possible, on the other hand, to call a COBOL program at the most external level from a PASCAL+ compilation unit. It is not possible to call COBOL routines from a PASCAL+ main program identified by the keyword PROGRAM.

The only requirement is that a standard Pascal+ routine is included, and that during the link a procedure declaration from the COBOL program containing the "EXTERNAL" directive is present; refer to the chapter "COMMUNICATION BETWEEN PROGRAMS" in the manual COBOL - Reference Manual.

PASCAL+ requires that any routine used is included during the interface link, and not by the PASCAL+ library; refer to the manual PASCAL+ Program Preparation and Execution.

5. COBOL COMPATIBILITY

COMPATIBILITY BETWEEN COBOL AND ICE COBOL

The sources of incompatibility can be identified in three areas:

- At the source code level where the syntax of certain source statements and the Reserved words lists differ.
- At execution level where the execution of certain statements differs.
- In the extensions to COBOL offered in the ICE COBOL implementation.

INCOMPATIBILITIES AT SOURCE LEVEL

- Source code formats.
  - . ICE COBOL permits two source code formats TEXT and CARD, COBOL permits only the latter. Files may be converted using the system editor.
- Reserved Word lists.
  - . COBOL has the following as reserved words ICE COBOL does not:

    ALSO, ANY, AREAS, ASCENDING, BINARY, BOTTOM, CF, CH, CODE, CODE-SET, COMP-1, COMP-2, COMP-4, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-4, CONSOLE, CONTROL, CONTROLS, CONVERTING, COUNT, DE, DEBUG-CONTENTS, DEBUG-ITEM, DEBUG-LINE, DEBUG-NAME, DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3, DEBUGGING, DELIMITED, DELIMITER, DESCENDING, DETAIL, DOUBLE-REAL, EDIT, END-ACCEPT, END-ADD, END-CALL, END-COMPUTE, END-DELETE, END-DIVIDE, END-EDIT, END-EVALUATE, END-IF, END-MULTIPLY, END-OF-PAGE, END-PERFORM, END-READ, END-RECEIVE, END-RETURN, END-REWRITE, END-SEARCH, END-START, END-STRING, END-SUBTRACT, END-UNSTRING, END-WRITE, EOP, EVALUATE, EXTERNAL, FALSE, FINAL, FIRST, FOOTING, FRONT-FEED, GENERATE, GLOBAL, GROUP, HEADING, INDICATE, INITIATE, LAST, LENGTH, LIMIT, LIMITS, LINAGE, LINAGE-COUNTER, LINE-COUNTER, MERGE, MESSAGE, MULTIPLE, NATIVE, OPTIONAL, ORDER, OTHER, PACKED-DECIMAL, PAGE-COUNTER, PF, PH, POINTER, POSITION, PRIMARY, PRINTING, PROCEDURES, RD, REEL, REFERENCES, RELEASE, REMOVAL, RENAMES, REPORT, REPORTING, REPORTS, RESERVE, RESET, RETURN, REVERSED, REWIND, RF, RH, SD, SEARCH, SEGMENT-LIMIT, SEQUENCE, SINGLE-REEL, SORT, SORT-MERGE, SOURCE, STANDARD-1, STRING, SUM, SUPPRESS, TAPE, TERMINATE, TOP, TRUE, TYPE, UNIT, UNSTRING, UPON, VALUES, WORD-BINARY,

    and the following division dependent system names:

M-30, M-40, REPORT-FILE, SYSPRINT-1, SYSPRINT-2, SYSPRINT-3, SYSPRINT-4, TEXT-FILE, ALLOC-UNIT, FILE-SIZE, WRITE-TYPE, LINE-PARAM.

The use of any of the following words from the ANSI standard reserved word list though not implemented result in the output of a warning message by the COBOL compiler:

CLOCK-UNITS, COMMON, COMMUNICATION, CD, DESTINATION, DISABLE, EGI, EMI, ENABLE, ESI, EVERY, QUEUE, RECEIVE, REPLACE, RERUN, SEGMENT, SEND, SUB-QUEUE-1, SUB-QUEUE-2, SUB-QUEUE-3, SYMBOLIC, TABLE, TERMINAL, TEXT.

The syntax of the following statements differs. In COBOL :

- The COPY statement permits a file-name and the path is specified using the OF clause, in ICE COBOL a text-name which may be a path-name is permitted.

- The Computer-name must be M30 or M40 in ICE COBOL it is treated as a comment.

- The MEMORY SIZE clause is not implemented, in ICE COBOL it is permitted for documentation.

- The OPEN statement in ICE COBOL may be specified with the EXCLUSIVE clause. The same function is obtained in COBOL using the WITH LOCK clause, the EXCLUSIVE clause is not permitted.

- The reserved word RECORD in the UNLOCK statement is required but is optional in ICE COBOL.

- The combined arithmetic operators ie. '>=' are not permitted but they are in ICE COBOL.

- The SYNCHRONIZED clause affects the memory allocation for an item in ICE COBOL this clause is treated as a comment. 110 The SAME AREA clause is treated as a comment, in ICE COBOL this clause is treated as if it were SAME RECORD AREA.

- The clauses VALUE OF FILE-SIZE and VALUE OF ALLOC-UNIT in the File Description Entry are the equivalent of the ICE COBOL clauses INITIAL-SIZE and INCREMENTAL-SIZE of the FILE-CONTROL paragraph.

- The ASSIGN clause differs in that:

  COBOL does not permit a data name as an argument.

  COBOL has nothing which corresponds to the ICE COBOL DISPLAY or KEYBOARD.

  COBOL assumes the argument DISK as default this must be specified for ICE.

47 CONTINUATION LINE MUST START IN MARGIN B

48 UNNECESSARY HYPHEN IN INDICATOR AREA

49 MISSING CLOSING QUOTE; ASSUMED

50 MISSING QUOTE TO CONTINUE A NONNUMERIC LITERAL; ASSUMED

51 WRONG SEQUENCE NUMBERING

52 ILLEGAL CHARACTER IN INDICATOR AREA; LINE IGNORED

53 ILLEGAL LOWER CASE LETTER IN COLUMN "nn"

54 ILLEGAL LOWER CASE LETTER IN HEX CONSTANT IN COLUMN "nn"

55 LINE IS LONGER THAN 80 CHARACTERS; TRUNCATED

56 MISSING HYPHEN IN INDICATOR AREA; ASSUMED

57 DEBUGGING LINE MUST NOT BE A CONTINUATION LINE

58 DEBUGGING LINE MUST NOT BE A CONTINUED LINE

59 ONLY DIGIT ALLOWED WITHIN FIRST SIX COLUMNS

60 ILLEGAL CHARACTER IN HEX CONSTANT IN COLUMN "nn"

61 PICTURE MISSING; PIC X ASSUMED

62 UNKNOWN PICTURE CHARACTER IN COLUMN "nn"; IGNORED

63 PICTURE STRING LONGER THAN 30 CHARACTERS; TRUNCATED

64 PICTURE BADLY FORMED IN POSITION "nn"; PIC X ASSUMED

65 CHARACTER IN POSITION "nn" MAY NOT BE REPEATED

66 MISSING OR ILLEGAL REPLICATION COUNT IN POSITION "nn"

67 RIGHT PARENTHESIS MISSING IN POSITION "nn"

68 REPLICATION COUNT MAY NOT BE ZERO IN POSITION "nn"

69 FIXED INSERTION INCOMPLETE; MISSING R AFTER POSITION "nn"

70 FIXED INSERTION INCOMPLETE; MISSING B AFTER POSITION "nn"

71 FIXED INSERTION INCOMPLETE; MISSING C BEFORE POSITION "nn"

72 POSITION "nn" MAKES PICTURE CATEGORY ILLEGAL; PIC X ASSUMED

73 ALPHANUMERIC EDITED PICTURES MUST CONTAIN A OR X; PIC X ASSUMED

74 DUPLICATED DECIMAL POINT IN POSITION "nn"; PIC 9 ASSUMED

75 SIZE REPRESENTED BY PICTURE IS GREATER THAN 32766; PIC X ASSUMED

76 SIZE REPRESENTED BY PICTURE IS GREATER THAN 32766; PIC 9 ASSUMED

77 SCALE OF PICTURE IS GREATER THAN 18; PIC 9 ASSUMED

78 TOO MANY P's IN PICTURE; PIC 9 ASSUMED

79 ILLEGAL PICTURE CAPACITY; PIC 9 ASSUMED

80 ILLEGAL FLOATING INSERTION SYMBOL IN POSITION "nn"; PIC 9 ASSUMED

81 ILLEGAL CHARACTER IN POSITION "nn"; PIC 9 ASSUMED

82 LIBRARY-TEXT IN LINE "nnn": CONTINUATION LINE MUST START IN MARGIN B

83 LIBRARY-TEXT IN LINE "nnn": UNNECESSARY HYPHEN IN INDICATOR AREA

84 LIBRARY-TEXT IN LINE "nnn": MISSING CLOSING QUOTE; ASSUMED

85 LIBRARY-TEXT IN LINE "nnn": MISSING QUOTE TO CONTINUE A NONNUMERIC LITERAL; ASSUMED

86 LIBRARY-TEXT IN LINE "nnn": ILLEGAL CHARACTER IN INDICATOR AREA; LINE IGNORED

87 LIBRARY-TEXT IN LINE "nnn": MISSING HYPHEN IN INDICATOR AREA; ASSUMED

88 LIBRARY-TEXT IN LINE "nnn": HEX CONSTANT MUST END WITH 2 STRING DELIMITERS; ASSUMED

89 LIBRARY-TEXT IN LINE "nnn": DEBUGGING LINE MUST NOT BE A CONTINUATION LINE

90 LIBRARY-TEXT IN LINE "nnn": DEBUGGING LINE MUST NOT BE A CONTINUED LINE

91 COPY FILE "name" DOES NOT EXIST

92 MISSING PERIOD AFTER PROGRAM-ID; ASSUMED

93 CHARACTER - NOT ALLOWED WITHIN PROGRAM NAME

94 ATTEMPT TO RECOVER AT "name"

95 NESTING OF COPY STATEMENTS NOT ALLOWED

96 TOO MANY LINES OF SOURCE PROGRAM

101 SIZE REPRESENTED BY PICTURE IS GREATER THAN 16383 ; PIC X ASSUMED

102 POSITION "number" MAKES PICTURE CATEGORY ILLEGAL ; PIC X ASSUMED

103 ILLEGAL CHARACTER IN KUTEN CONSTANT IN COLUMN "number"

104 FLOATING LITERAL EXPONENT LONGER THAN 3 DIGITS IN COLUMN "number" ; TRUNCATED

105 FLOATING LITERAL EXPONENT MISSING IN COLUMN "number" ; DEFAULT ASSUMED

106 FLOATING LITERAL MANTISSA LONGER THAN 15 DIGITS IN COLUMN "number" ; TRUNCATED

108 ILLEGAL COMMENT(S) BEFORE IDENTIFICATION DIVISION

109 ILLEGAL COMMA AT END OF PICTURE; SEPARATOR COMMA ASSUMED

110 IMPROPER STRING DELIMITER IN COLUMN "number"; DELIMITER CHANGED

2001 TEXT NOT EXPECTED TO BEGIN IN MARGIN A

2002 DUPLICATED "reserved word" CLAUSE; PREVIOUS DISCARDED

2003 DUPLICATED "reserved word" CLAUSE; DISCARDED

2004 "reserved word" CLAUSE NOT ALLOWED IN THIS CONTEXT; DISCARDED

2005 DIVISION EXPECTED; FOUND "name"; DIVISION ASSUMED

2006 PERIOD EXPECTED; FOUND "character"; PERIOD ASSUMED

2007 SECTION EXPECTED; FOUND "name"; SECTION ASSUMED

2008 TEXT EXPECTED TO BEGIN IN MARGIN A

2009 LEVEL-NUMBER MAY NOT BE SIGNED; SIGN IGNORED

2010 EXCESS ZEROS IN LEVEL-NUMBER; IGNORED

2011 EXCESS QUALIFIER "name"; DISCARDED

2012 MORE THAN 3 SUBSCRIPTS FOR "name"; DISCARDED

2013 INTEGER EXPECTED; FOUND "text"

2014 ILLEGAL LEVEL-NUMBER; 01 ASSUMED

2015 ILLEGAL LEVEL-NUMBER; 49 ASSUMED

2016 SEGMENT-NUMBER MAY NOT BE SIGNED; SIGN IGNORED

2017 EXCESS ZEROS IN SEGMENT-NUMBER; IGNORED

2018 ILLEGAL SEGMENT-NUMBER

2019 ZERO ILLEGAL IN SUBSCRIPT; 1 ASSUMED

2020 RIGHT PARENTHESIS EXPECTED; FOUND "character"; PARENTHESIS ASSUMED

2021 QUALIFIER-NAME EXPECTED; FOUND "text"; OF/IN DISCARDED

2022 INTEGER EXPECTED IN SUBSCRIPT; FOUND "text"; 1 ASSUMED

2023 DUPLICATED BLOCK CONTAINS CLAUSE; PREVIOUS DISCARDED

2024 DUPLICATED EXTERNAL/GLOBAL CLAUSE; PREVIOUS DISCARDED

2025 DUPLICATED DATA RECORD CLAUSE; PREVIOUS DISCARDED

2026 UPPER & LOWER BOUNDS OF BLOCK CONTAINS INVERTED

2027 UPPER & LOWER BOUNDS OF RECORD CONTAINS INVERTED

2028 DUPLICATED RECORD CONTAINS CLAUSE; PREVIOUS DISCARDED

2029 DUPLICATED FILE STATUS CLAUSE; PREVIOUS DISCARDED

2030 DUPLICATED RECORD KEY CLAUSE; PREVIOUS DISCARDED

2031 DUPLICATED ALTERNATE RECORD CLAUSE; PREVIOUS DISCARDED

2032 DUPLICATED BLANK WHEN ZERO CLAUSE; DISCARDED

2033 BLANK WHEN ZERO NOT ALLOWED IN THIS CONTEXT; DISCARDED

2034 DEPENDING ON NOT ALLOWED IN THIS CONTEXT; DISCARDED

2035 RECORD KEY CLAUSE NOT ALLOWED IN THIS CONTEXT; DISCARDED

2036 RECORD CONTAINS CLAUSE NOT ALLOWED IN THIS CONTEXT; DISCARDED

2037 LINAGE CLAUSE NOT ALLOWED FOR REPORT FILE; DISCARDED

2038 DATA RECORDS CLAUSE NOT ALLOWED FOR REPORT FILE; DISCARDED

2039 BLOCK CONTAINS CLAUSE NOT ALLOWED FOR REPORT FILE; DISCARDED

2040 RELATIVE KEY CLAUSE NOT ALLOWED IN THIS CONTEXT; DISCARDED

2041 ALTERNATE RECORD CLAUSE NOT ALLOWED IN THIS CONTEXT; DISCARDED

2042 NUMERIC VALUE NOT ALLOWED IN THIS CONTEXT; SPACE ASSUMED

2043 NONNUMERIC VALUE NOT ALLOWED IN THIS CONTEXT; ZERO ASSUMED

2044 LINE OR USAGE CLAUSE REQUIRED FOR NESTED REPORT GROUP

2045 SIGN SEPARATE ASSUMED

2046 OCCURS UPPER-BOUND MUST BE POSITIVE & NON-ZERO; 1 ASSUMED

2047 OCCURS LOWER-BOUND MUST BE NON-NEGATIVE; ZERO ASSUMED

2048 OCCURS LOWER- & UPPER-BOUNDS EXCHANGED

2049 USAGE CHANGED TO DISPLAY

2050 SEQUENTIAL ACCESS MODE REQUIRED IN THIS CONTEXT; ASSUMED

2051 ACCESS MODE CHANGED TO SEQUENTIAL

2052 RELATIVE KEY CLAUSE REQUIRED IN THIS CONTEXT

2053 ILLEGAL ALL IN NUMERIC ALPHABET, OR CLASS CONTEXT; IGNORED

2054 VALUE OF LITERAL BEYOND CAPACITY OF ALPHABET OR CLASS

2055 LITERAL SPECIFIED MORE THAN ONCE; IGNORED

2056 NONNUMERIC LITERAL "item" TRUNCATED TO 1 CHARACTER

2057 CODE-LITERAL NOT CHARACTER-STRING; BLANKS ASSUMED

2058 CODE-LITERAL NOT 2 CHARACTERS; BLANKS ASSUMED

2059 DUPLICATED NEXT GROUP CLAUSE; PREVIOUS DISCARDED

2060 DUPLICATED GROUP INDICATE CLAUSE; DISCARDED

2061 DUPLICATED SUM CLAUSE; ALL OCCURRENCES COMBINED

2062 DATA-NAME EXPECTED; FOUND "name"; FILLER ASSUMED

2063 MODE EXPECTED; FOUND "name"; MODE ASSUMED

2064 INTEGER EXPECTED; FOUND "text"; 1 ASSUMED

2065 COMMA EXPECTED; FOUND "text"; COMMA ASSUMED

2066 FILE EXPECTED; FOUND "text"; FILE ASSUMED

2067 ZERO EXPECTED; FOUND "text"; ZERO ASSUMED

2068 PAGE EXPECTED; FOUND "text"; PAGE ASSUMED

2069 GROUP EXPECTED; FOUND "text"; GROUP ASSUMED

2070 NUMERIC LITERAL EXPECTED; FOUND "text"; 1 ASSUMED

2071 NONNUMERIC LITERAL EXPECTED; FOUND "text"

2072 TEXT-FILE NAME EXPECTED; FOUND "text"; SYSPRINT1 ASSUMED

2073 LIST OF FILE-NAMES EXPECTED; FOUND "text"

2074 "name" NOT LEGAL IN ASSIGN CLAUSE; ASSIGN CLAUSE DISCARDED

2075 "text" IMPROPER ORGANIZATION TYPE; SEQUENTIAL ASSUMED

2076 "text" IMPROPER ACCESS TYPE; SEQUENTIAL ASSUMED

2077 STANDARD OR OMITTED EXPECTED; FOUND "text"; STANDARD ASSUMED

2078 QUALIFIED DATA-NAME EXPECTED; FOUND "name"; FILLER ASSUMED

2079 DATA-NAME OR NUMBER EXPECTED; FOUND "name"; 1 ASSUMED

2080 RENAMES EXPECTED; FOUND "text"

2081 DEPENDING EXPECTED; FOUND "text"; DEPENDING ASSUMED

2082 VALUE IS EXPECTED; FOUND "text"; VALUE IS ASSUMED

2083 LITERAL EXPECTED; FOUND "text"; DEFAULT ASSUMED

2084 RECORD EXPECTED; FOUND "text"

2085 DISPLAY EXPECTED; FOUND "text"; DISPLAY ASSUMED

2086 HEADING OR FOOTING EXPECTED; FOUND "text"

2087 IDENTIFIER EXPECTED; FOUND "text"; FILLER ASSUMED

2088 DATA-NAME OR FINAL EXPECTED; FOUND "text"

2089 IDENTIFIER OR LITERAL EXPECTED; FOUND "text"

2090 DETAIL EXPECTED; FOUND "text"; DETAIL ASSUMED

2091 UNEXPECTED EOF

2092 SYNTAX ERROR DETECTED AT "name"

2093 ATTEMPT TO RECOVER AT "name"

2094 UNRECOGNIZED TEXT DISCARDED STARTING AT "text"

2095 PICTURE REQUIRED FOR ELEMENTARY REPORT ENTRY; PIC X ASSUMED

2096 "name" ILLEGAL COMPUTER-NAME

2097 IMPROPER USE OF IS/ARE

2098 ILLEGAL SWITCH INTEGER VALUE; "character" ASSUMED

2099 ILLEGAL LINE INTEGER VALUE; "character" ASSUMED

2100 INTEGER MAY NOT BE SIGNED; SIGN IGNORED

2101 NUMERIC PICTURE IN SCREEN SECTION MAY NOT BEGIN WITH P

2102 VALUE ILLEGAL WITH USING, FROM, OR TO PHRASE; DISCARDED

2103 VALUE, USING, FROM, OR TO MUST BE PRESENT; VALUE ZERO ASSUMED

2104 VALUE, USING, FROM, OR TO MUST BE PRESENT; VALUE SPACE ASSUMED

2105 FILE/ALLOC SIZE MUST BE UNSIGNED; SIGN IGNORED

2106 FILE/ALLOC SIZE MUST BE INTEGER; DISCARDED

2107 FILE/ALLOC SIZE TOO BIG; DISCARDED

2108 RECORDS EXPECTED; FOUND RECORD; RECORDS ASSUMED

2109 "name" IMPROPER FORMED PATHNAME

2110 PATHNAME "string" LONGER THAN SIXTY CHARACTERS

2111 PATHNAME "string" MUST BEGIN WITH /

2112 EXTRA ALTERNATE KEYS; IGNORED

2113 DUPLICATED VALUE OF CLAUSE; DISCARDED

2114 USER-PRINTER FOUND; SYSPRINT1 ASSUMED

2115 MISSING RECORD DESCRIPTION ENTRY; 01 FILLER PIC X ASSUMED

2116 ILLEGAL PAGE-SIZE VALUE; DEFAULT ASSUMED

2117 ILLEGAL SPLIT-STRATEGY VALUE; DEFAULT ASSUMED

2118 ILLEGAL TIME-OUT VALUE; DEFAULT ASSUMED

2120 ILLEGAL LITERAL IN VALUE CLAUSE; DEFAULT ASSUMED

2122 ILLEGAL BOX VALUE; ZERO ASSUMED

2123 ILLEGAL COLUMN INTEGER VALUE; 1 ASSUMED

2124 MISSING TYPE CLAUSE; REPORT GROUP DISCARDED

2125 SECTION OUT OF ORDER

3001 RECORD CONTAINS INTEGER LESS THAN ACTUAL RECORD SIZE OF "name", IGNORED

3002 DUPLICATE SELECT FOR THE SAME FD; DISCARDED

3003 SIZE OF LINKAGE SECTION BIGGER THAN 65535 BYTES

3004 DUPLICATE EXTERNAL NAME; EXTERNAL IGNORED

3005 SAME RECORD AREA ILLEGAL FOR EXTERNAL FILE; IGNORED

3006 NAME "name" : NAMING CONFLICT WITH INDEX-NAME

3007 QUALIFIER "name" UNKNOWN IN CONTEXT; QUALIFIER IGNORED

3008 ILLEGAL LEVEL-NUMBER IN RENAMES OBJECT

3009 RENAMES OBJECT CONTAINS VARIABLE OCCURRENCE TABLE(S)

3010 RENAMES OBJECT CANNOT BE (PART OF) A TABLE

3011 UNKNOWN RENAMES OBJECT

3012 VARIABLE OCCURRENCE ITEMS BETWEEN RENAMES OBJECTS; THRU DISCARDED

3013 THRU OBJECT SUBORDINATE TO RENAMES OBJECT; THRU DISCARDED

3014 THRU OBJECT PRECEEDING RENAMES OBJECT; THRU DISCARDED

3015 RENAMES AND THRU OBJECTS NOT PART OF SAME RECORD; THRU DISCARDED

3016 RENAMES/THRU OBJECT NOT PART OF CURRENT RECORD

3017 LEVEL OF TABLE NESTING GREATER THAN THREE; OCCURS DISCARDED

3018 NESTED VARIABLE OCCURRENCE TABLE; DEPENDING ON DISCARDED

3019 VARIABLE OCCURRENCE TABLE IN REDEFINITION; DEPENDING ON DISCARDED

3020 USAGE OVERRIDES GROUP USAGE

3021 ILLEGAL GROUP USAGE; DISPLAY ASSUMED

3022 SIGN OVERRIDES GROUP SIGN

3023 REDEFINES OBJECT UNKNOWN OR ILLEGAL; "name" ASSUMED

3024 REDEFINES OBJECT UNKNOWN OR ILLEGAL; REDEFINES DISCARDED

3025 ILLEGAL GROUP SIGN; DISCARDED

3026 ILLEGAL GROUP VALUE; DISCARDED

3079 SIZE OF VALUE LITERAL GREATER THAN ITEM SIZE; TRUNCATED

3080 ILLEGALLY SIGNED LITERAL; SIGN DISCARDED

3081 HIERARCHY MORE THAN THREE LEVELS DEEP

3082 TOO MANY SAME RECORD AREA CLAUSES

3083 RECORD SIZE LARGER THAN 32766 BYTES

3084 FULL/REQUIRED OVERRIDES GROUP'S REQUIRED/FULL

3085 OVERLAPPING FIELDS

3086 FIELD OUTSIDE SCREEN LIMITS

3087 "name" MUST BE GROUP; USING DISCARDED

3088 "name" MUST BE GROUP; FROM DISCARDED

3089 "name" MUST BE GROUP; TO DISCARDED

3090 "name" MUST BE GROUP; VALUE DISCARDED

3091 "name" MUST BE GROUP; COLUMN DISCARDED

3092 LINE NOT RELATIVE; LINE PLUS 1 ASSUMED

3093 "name" MUST BE GROUP; BLANK LINE DISCARDED

3094 SECURE/AUTO/REQUIRED/FULL DISCARDED FOR OUTPUT FIELD

3095 BLANK SCREEN DISCARDED FOR INPUT FIELD

3096 "name" MUST BE ELEMENTARY; PIC X VALUE SPACE ASSUMED

3097 VALUE OF "name" NOT ALLOWED BY ORGANIZATION

3098 "name" MUST BE DECLARED TEXT-FILE; ASSUMED

3099 ILLEGAL SEQUENCE OF LINE NUMBER IN GROUP "nn"

3100 ILLEGAL NEXT GROUP CLAUSE IN GROUP "nn"

3101 LINE NUMBER OUT OF LIMITS IN GROUP "nn"

3102 OVERFLOW ON NEXT GROUP CLAUSE OF GROUP "nn"

3103 VALUE OF "name' LESS THAN RECORD LENGTH; DEFAULT ASSUMED

3104 SYNC RIGHT ASSUMED

3105 ILLEGAL ORGANIZATION AND ACCESS; SEQUENTIAL ASSUMED

3106 "nome" CANNOT BE ASSCIATED WITH CONDITION-NAME; CONDITION-NAME DISCARDED

3107 RECORD CONTAINS INTEGER LARGER THAN ACTUAL RECORD SIZE OF "nn"; IGNORED

3108 BLOCK CONTAINS INTEGER LESS THAN ACTUAL RECORD SIZE OF "nn"; IGNORED

3109 ILLEGAL NEXT GROUP CLAUSE IN GROUP "nn"; DISCARDED

3010 ILLEGAL GROUP INDICATED CLAUSE IN GROUP "nn"; DISCARD

3111 JUSTIFIED CANNOT BE SPECIFIED WITH FULL CLAUSE; IGNORED

4001 AMBIGUOUS REFERENCE TO "name"; LAST OCCURRENCE USED

4002 UNDEFINED USER NAME "name"

4003 "name" MUST BE A DATA NAME

4004 "name" MUST BE A FILE

4005 "name" MUST BE A REPORT DEFN

4006 "name" MUST BE A PROCEDURE

4007 CONDITION NAME "name" USED ILLEGALLY

4008 "name" MUST BE A SCREEN NAME

4009 REPORT NAME "name" MUST BE A SUM COUNTER TO BE REFERENCED AS A DATA NAME

4010 "name" MUST BE A REPORT GROUP

4011 RELATIVE KEY "name" IN FILE RECORD

4012 RELATIVE KEY "name" MUST HAVE NO SCALE

4013 RELATIVE KEY "name" HAS IMPROPER DATA TYPE

4014 PRIMARY/ALTERNATE KEY "name" NOT CONTAINED IN ALL RECORDS

4015 PRIMARY/ALTERNATE KEY "name" NOT IN FILE RECORD

4016 PRIMARY/ALTERNATE KEY "name" IS VARIABLE LENGTH

4017 PRIMARY/ALTERNATE KEY "name" NOT ALPHANUMERIC

4018 IMPROPER DATA TYPE OR SIZE FOR FILE STATUS "name"

4019 FILE STATUS "name" DEFINED IN ILLEGAL SECTION

4020 DEPENDING "name" MUST BE EXTERNAL

4021 DEPENDING "name" MUST BE GLOBAL

4022 DEPENDING "name" NOT IN SAME PROG

4023 DEPENDING "name" NOT WITHIN FILE RECORD

4024 DEPENDING "name" DEFINED WITHIN TABLE

4025 DEPENDING "name" NOT INTEGER

4026 DEPENDING "name" NOT PROPER DATA TYPE

4027 DATA NAME "name" NOT FIRST KEY

4028 TABLE KEY "name" NOT AT SAME NESTING LEVEL

4029 TABLE KEY "name" NOT IN TABLE

4030 DATA RECORDS NAME "name" NOT RECORD

4031 LINAGE/VALUE OF NAME "name" NOT INTEGER, DEFAULT USED

4032 LINAGE/VALUE OF NAME "name" NOT PROPER DATA TYPE, DEFAULT USED

4033 LINAGE/VALUE OF NAME "name" NOT EXTERNAL

4034 UPON NAME "name" NOT DETAIL REPORT GROUP

4035 RESET CONTROL NAME "name" IS LOWER LEVEL CONTROL

4036 RESET NAME "name" NOT A CONTROL

4037 CONTROL NAME "name" IS VARIABLE LENGTH

4038 CONTROL NAME "name" DEFINED IN REPORT SECTION

4039 DUPLICATE NAME "name" ON CONTROLS

4040 CONTROL NAME "name" REFERENCED TWICE AS HEADER

4041 CONTROL NAME "name" REFERENCED TWICE AS FOOTING

4042 "name" NOT DECLARED AS CONTROL ITEM

4043 MORE THAN 1 FILE USED FOR REPORT "name"

4044 SUM NAME "name" NOT NUMERIC

4045 SOURCE NAME "name" NOT IN CURRENT REPORT

4046 SUM IS PAGE COUNTER OR LINE COUNTER

4047 PRIMARY/ALTERNATE KEY "name" TOO LONG

4048 "name" NOT IN FILE OR WORKING-STORAGE

4049 SUBSCRIPT "name" IS SUM COUNTER

4050 SUBSCRIPT IS PAGE COUNTER OR LINE COUNTER

4051 SUBSCRIPT NAME "name" NOT INTEGER

4052 SUBSCRIPT NAME "name" NOT PROPER DATA TYPE

4053 SUBSCRIPT NAME "name" IS IN TABLE

4054 SUBSCRIPT ILLEGAL FOR "name"

4055 WRONG NUMBER OF SUBSCRIPTS FOR "name"

4056 2 KEYS WITH SAME STARTING ADDR FOR FILE "file name"

4057 NULL STATEMENT

4058 EXTRANEOUS ELSE

4059 MISSING "string"; ASSUMED

4060 TEXT NOT EXPECTED TO BEGIN IN MARGIN A

4061 DIVISION EXPECTED; FOUND "text"; DIVISION ASSUMED

4062 PERIOD EXPECTED; FOUND "character"; PERIOD ASSUMED

4063 TEXT EXPECTED TO BE IN MARGIN A

4064 EXCESS QUALIFIER "name" DISCARDED

4065 MORE THAN 3 SUBSCRIPTS ARE PRESENT FOR "name"

4066 DUPLICATE PARAMETER "name"

4067 SYNTAX ERROR DETECTED AT "name".

4068 UNRECOGNIZED TEXT DISCARDED STARTING AT "name"

4069 ATTEMPT TO RECOVER AT "name"

4070 BAD SEGMENT NUMBER "nn"

4071 EXCESSIVE NOTS

4072 INTEGER "name" MUST NOT BE SIGNED; SIGN IGNORED

4073 ZERO VALUE NOT ALLOWED IN THIS CONTEXT; 1 ASSUMED

4074 INTEGER "name" MUST NOT BE NEGATIVE; SIGN IGNORED

4075 INTEGER EXPECTED IN SUBSCRIPT; FOUND "name"; 1 ASSUMED

4076 RIGHT PARENTHESIS EXPECTED; FOUND "character"; PARENTHESIS ASSUMED

4077 QUALIFIER NAME EXPECTED; FOUND "name"; OF/IN DISCARDED

4078 MISSING DECLARATIVE STATEMENT

4079 MISSING SECTION DEFINITION; DUMMY CREATED

4080 MISSING PARAGRAPH DEFINITION; DUMMY CREATED

4081 ILLEGAL SECTION "name" SINCE STARTED W/PARAGRAPH

4082 ILLEGAL SECTION NUMBER FOR PARAGRAPH "name" IGNORED

4083 ILLEGAL ALL IN NUMERIC OR FIGURATIVE CONSTANT CONTEXT IGNORED

4084 LINKAGE ITEM NOT REFERENCED IN USING

4085 MORE THAN 25 NESTED PERFORM STATEMENTS

4086 KEY EXPECTED; FOUND "name"; KEY ASSUMED

4087 NUMERIC LITERAL MUST BE INTEGER; SCALE IGNORED

4088 RUN EXPECTED; FOUND "name"; RUN ASSUMED

4089 GO DEPENDING NAME "name" NOT PROPER DATA TYPE

4090 GO DEPENDING NAME "name" NOT INTEGER

4091 DEPENDING EXPECTED; FOUND "name"

4092 USING VAR "name" NOT DEFINED IN LINKAGE SECTION

4093 USING VAR "name" NOT DECLARED AS 01 OR 77

4094 USING VAR "name" CONTAINS REDEFINES

4095 CODE NOT PRESENT FOR REPORT "name"; ONE ASSUMED

4096 CODE IGNORED FOR REPORT "name"

4097 NOT ENOUGH FIELDS BEFORE GIVING; TO ASSUMED

4098 ROUND BEFORE GIVING; GIVING IGNORED

4099 DECLARATIVES EXPECTED; FOUND "name"; DECLARATIVES ASSUMED

4100 TIMES EXPECTED; FOUND "name"; TIMES ASSUMED

4101 "name" MUST BE A DATA NAME, NOT INDEX-NAME

4102 "name" MUST BE AN INDEX-NAME OR DATA NAME

4103 INDEX NAMES MAY NOT BE QUALIFIED

4104 "name" MUST BE A CONDITION NAME

4105 "name" MUST BE A MNEMONIC NAME

4106 BY EXPECTED; FOUND "name"; BY ASSUMED

4107 PARAMETER "name" DECLARED IN RW SECTION

4108 PARAMETER "name" IS BINARY AND NOT SYNCHRONIZED

4109 PARAMETER "name" IS NOT ALIGNED ON WORD BOUNDARY

4110 ILLEGAL USE OF USAGE IS INDEX ITEM "name"

4111 CANCEL LITERAL "name" IS AN INTERNAL PROGRAM NAME

4112 RECURSIVE PROGRAM IS ILLEGAL

4113 MNEMONIC COND NAME ILLEGAL IN THIS CONTEXT

4114 MNEMONIC NAME "name" MUST BE CONSOLE

4115 MNEMONIC NAME "name" MUST BE SWITCH TYPE

4116 MNEMONIC NAME "name" MUST BE LINE TYPE

4117 SENTENCE EXPECTED; FOUND "name"; SENTENCE ASSUMED

4118 NEXT SENTENCE IN ILLEGAL CONTEXT

4119 INCOMPLETE CONDITIONAL PHRASE; ASSUMED COMPLETE

4120 STATEMENT MAY NOT BECOME CONDITIONAL

4121 MISSING AT END IN RETURN STATEMENT

4122 MISMATCHED CONDITIONAL STATEMENT RESERVED WORD "name"

4123 MISMATCHED DELIMITED SCOPE RESERVED WORD "name"

4124 NO VARIABLE IS REFERENCED IN SIMPLE CONDITION

4125 IMPROPER USE OF CLASS CONDITION

4126 IMPROPER USE OF CONDITIONAL OPERATOR

4127 CONDITIONAL EXPRESSION IN ARITHMETIC CONTEXT

4128 ILLEGAL ABBREVIATED CONDITION

4129 ARITHMETIC EXPRESSION IN CONDITIONAL CONTEXT

4130 EXIT MUST BE ONLY STATEMENT IN SENTENCE

4131 EXIT SENTENCE MUST BE ONLY SENTENCE IN PARAGRAPH

4132 EXIT PROGRAM NOT LAST STATEMENT IN SEQUENCE

4133 INVALID KEY CLAUSE ILLEGAL FOR FILE "file name"

4134 SEQUENTIAL FILE "file name" USED IN LOCK, START, OR UNLOCK STM

4135 "file name" MUST HAVE INDEXED ORGANIZATION; ON CLAUSE IGNORED

4136 REVERSED ILLEGAL FOR FILE "file name"

4137 OPEN I-O AND REWRITE ILLEGAL FOR TEXT FILE "file name"

4138 NEXT/PREVIOUS NOT ALLOWED FOR FILE "file name"

4139 LOCK PHRASE ILLEGAL FOR SEQUENTIAL FILE "file name"

4140 BEFORE/AFTER NOT ALLOWED FOR FILE "file name"

4141 FILE "file name" NOT DEFINED WITH A RELATIVE KEY PHRASE

4142 ILLEGAL KEY ON READ

4143 ILLEGAL KEY FOR FILE "file name"

4144 FILE "file name" HAS IMPROPER ACCESS FOR OPEN EXTEND

4145 FILE "file name" MAY NOT HAVE RANDOM ACCESS

4146 KEY IS PRIMARY KEY FOR FILE "file name"

4147 KEY IS NOT ALTERNATE KEY FOR FILE "file name"

4148 NOT RELATIVE KEY FOR FILE "file name"

4149 NOT KEY FOR INDEXED FILE "file name"

4150 BEFORE/AFTER ID "name" NOT INTEGER

4151 BEFORE/AFTER ID "name" NOT PROPER DATA TYPE

4152 AT END ILLEGAL FOR FILE "file name"

4153 END-OF-PAGE ILLEGAL FOR FILE "file name"

4154 "name" DECLARATIVE ALREADY PRESENT IN PROGRAM

4155 DECLARATIVE FOR "name" ALREADY DEFINED

4156 DECLARATIVE FOR "name" NOT IN SAME PROGRAM AS DECLARATION

4157 GLOBAL NOT ALLOWED IN THIS CONTEXT; IGNORED

4158 OPEN INPUT/EXTEND ILLEGAL FOR PRINTER FILE "file name"

4159 OPEN EXTEND ILLEGAL FOR LINAGE FILE "file name"

4160 NO DELIMITED PHRASE: COUNT/DELIMITER PHRASE IGNORED

4161 "name" NOT A RECORD IN FILE SECTION

4162 RECORD OF FILE MAY NOT SHARE STORAGE SPACE WITH FROM/INTO IDENTIFIER

4163 "file name" MUST HAVE SEQUENTIAL ORGANIZATION; CLAUSE IGNORED

4164 DELETE FOR SEQUENTIAL FILE "file name" IS ILLEGAL

4165 NO ADVANCING ILLEGAL ON SCREEN DISPLAY

4166 "name" MUST HAVE TABLE KEYS

4167 "name" MUST HAVE INDEXED BY CLAUSE

4168 "name" MUST HAVE OCCURS CLAUSE

4169 REFERENCE TO IMPROPERLY DEFINED CONDITION-NAME "name"

4170 SORT FILE/RECORD "name" NOT ALLOWED IN THIS CONTEXT

4171 "name" MUST BE A SORT FILE/RECORD

4172 SORT/MERGE KEY "name" CONTAINS OCCURS OR IS VARIABLE LENGTH

4173 INDEX-NAME "name" NOT IN TABLE

4174 NO LOCK NOT ALLOWED FOR FILE "name"; DISCARDED

4175 MISSING OPERAND

4176 ON EXCEPTION/OVERFLOW NOT EXECUTABLE

4177 EXIT PROGRAM IN DECLARATIVE IGNORED

4178 ALL TABLE KEYS PREVIOUS TO "name" HAVE NOT BEEN SPECIFIED

4179 "name" NOT KEY OF SEARCH TABLE

4180 TABLE KEY "name" HAS ILLEGAL SUBSCRIPT

4181 CONDITION-NAME "name" HAS MORE THAN ONE VALUE

4182 MISSING ON EXCEPTION IN ACCEPT OR EDIT STATEMENT

4183 FLOATING-POINT LITERAL IN ILLEGAL CONTEXT

4184 ASSIGN NAME "name" NOT PROPER DATA TYPE

4185 ASSIGN NAME "name" NOT EXTERNAL

4186 EXTRANEOUS WHEN DISCARDED

4187 SORT KEY "name" NOT IN SD RECORD

4188 SUPPRESS STATEMENT NOT IN REPORT DECLARATIVE

4189 MISSING DEFINITION OF APPROPRIATE TYPE FOR "name"

4190 NO FILE SPECIFIED FOR REPORT "name"

4191 ILLEGAL SPECIFICATION OF "name" IN GENERATE STATEMENT

4192 UNDEFINED SPECIAL NAME "name"

4193 ILLEGAL LITERAL IN DISPLAY SCREEN STATEMENT DISCARED

5001 ID-1 OR ID-2 CONDITIONAL DATA

5002 ID-1 OR ID-2 ILLEGAL LEVEL-NUMBER

5003 ID-1 AND ID-2 NOT GROUP ITEMS

5004 NO MATCHES FOUND

6001 "name" IS NOT AN 01 ENTRY IN THE DISPLAY STATEMENT - IGNORED

6002 "name" IS NOT IN THE SAME 01 ENTRY AS THE FIRST SCREEN NAME - IGNORED

6003 "name" IS REFERENCED MORE THAN ONCE IN DISPLAY STATEMENT - IGNORED

6004 FROM/USING SPECIFICATION IS INCOMPATIBLE WITH SCREEN ITEM "name"

6005 "name" DOES NOT CONTAIN A USING PHRASE OR FROM AND TO PHRASES

6006 "name" DOES NOT CONTAIN A TO OR USING PHRASE

6007 "name" IS NOT INTEGER IN INTEGER CONTEXT

6008 NO P'S ALLOWED IN PICTURE FOR "name"

6009 "name" MUST BE USAGE DISPLAY

6010 STRING/UNSTRING POINTER FIELD "name" TOO SMALL

6011 "name" MAY NOT BE EDITED

6012 ILLEGAL PRESENCE OF JUSTIFIED CLAUSE FOR "name"

6013 "name" IS NOT NUMERIC IN NUMERIC CONTEXT

6014 "name" MUST BE ALPHANUMERIC

6018 TOO MANY KEYS IN SORT/MERGE STATEMENT

6019 TOO MANY IDENTIFIERS IN DISPLAY STATEMENT

6020 "name" IS NOT A DETAIL GROUP

6021 "name" MAY BE ACCEPTED/DISPLAYED

6022 TOO MANY ELEMENTS IN SCREEN DEFINITION "name"

7001 TOO MANY PROC NAMES IN GO TO DEPENDING

7002 RECEIVING FIELD "name" IS INCOMPATIBLE WITH SENDING FIELD IN MOVE

7003 "name" HAS BAD DATA TYPE IN ARITHMETIC STATEMENT OR EXPRESSION

7004 INDEX "name" NOT ALLOWED IN ARITHMETIC EXPRESSION

7005 COMPOSITE IN ARITHMETIC STATEMENT IS TOO BIG

7006 "name" IS NOT INTEGER IN INTEGER CONTEXT

7007 "name" IS NOT NUMERIC IN NUMERIC CONTEXT

7008 CALL IDENTIFIER HAS WRONG DATA TYPE

7009 NUMBER OF PARAMETERS IN CALL AND PROCEDURE DIVISION HEADER DIFFERENT

7010 NO P'S ALLOWED IN PICTURE FOR "name"

7011 "name" MUST BE AT LEAST TWO DIGITS LONG

7012 DISPLAY ITEMS EXCEED SCREEN SIZE

7013 SUBJECT OF SIGN CONDITION MUST BE ARITHMETIC

7014 SUBJECT OF CLASS CONDITION MUST NOT BE ALPHABETIC

7015 SUBJECT OF CLASS CONDITION MUST NOT BE NUMERIC

7016 ILLEGAL COMPARE INVOLVING AN INDEX DATA ITEM

7017 ILLEGAL NONNUMERIC COMPARE

7018 BAD SET OPERAND "name"

7019 MORE THAN 25 NESTED PERFORMS

7020 PROCEDURE NAMES ARE IN DIFFERENT DECLARATIVE SECTIONS

7021 "name" MUST BE USAGE DISPLAY

7022 SIZES OF IDENTIFIERS/LITERALS NOT EQUAL IN INSPECT

7023 WRONG NUMBER OF PARAMETERS FOR VISA CALL

7024 ILLEGAL LITERAL IN FROM CLAUSE

7025 ILLEGAL LITERAL IN BY CLAUSE

7026 USE PROCEDURE OR CONDITIONAL STATEMENT REQUIRED IN THIS CONTEXT

7027 SORT/MERGE STATEMENT NOT ALLOWED IN DECLARATIVE

7028 SORT/MERGE STATEMENT NOT ALLOWED IN INPUT OR OUTPUT PROCEDURE

7029 RELEASE STATEMENT POSSIBLY NOT IN INPUT PROCEDURE

7030 RETURN STATEMENT POSSIBLY NOT IN OUTPUT PROCEDURE

7031 TOO MANY DECLARATIVE SECTIONS

7035 "name" IS NOT ALPHANUMERIC IN ALPHANUMERIC CONTEXT

7036 LINE-PARAM KEY NOT 4 CHARACTERS

7037 ILLEGAL SUBSCRIPT IN SEARCH VARYING IDENTIFIER

7038 REPEATED CHARACTER IN FIRST OPERAND OF INSPECT CONVERTING

7039 ILLEGAL ROUNDED FOR FLOATING ITEM IGNORED

7040 TOO MANY EVALUATE OBJECTS IN WHEN CLAUSE

7041 TOO FEW EVALUATE OBJECTS IN WHEN CLAUSE; ANY ASSUMED

7042 OBJECT NOT CONSISTENT WITH SUBJECT NUMBER "nn"

7043 TOO MANY EVALUATE SUBJECTS AND CONDITIONS IN EVALUATE STATEMENT(S)

7044 ILLEGAL USE OF USAGE IS INDEX ITEM "name"

7045 IMPROPER REFERENCE TO "name" IN GO STATEMENT

7046 IMPROPER REFERENCE TO NONDECLARATIVE PROCEDURE

9999 INTERNAL COMPILER ERROR xx xx

In this case the user should note the numbers displayed, the version of the compiler in use and call the nearest Olivetti assistance office.

B. RUN TIME MESSAGES

Two types of message are issued at run time in a COBOL environment:

- ERROR messages, comprising run time errors with a status code of I-O, and run time errors without I-O status code.

- Warning messages; these messages are self-explanatory; to maintain consistency the fixed parts of the messages are given below.

Information on the consequences and the means of recovery from run time errors is provided in the chapter "COBOL PROGRAM EXECUTION".

WARNING MESSAGES

There are two categories of warning messages, as follows:

1. Data input messages resulting from inconsistency between the data entered and that defined in the ACCEPT statement. The user must must input valid data.

   a) The following messages consists of two or three parts: the message text, the picture string of the data item if it is of category numeric, and a request to retype it.

      ILLEGAL NUMERIC DATA

      INTEGER SEPARATOR ERROR

      MORE THAN ONE DECIMAL SEPARATOR.POINT CHARACTER

      MORE THAN ONE SIGN CHARACTER

      NUMERIC STRING TOO LONG

      SIGN NOT ALLOWED FOR UNSIGNED FIELD

      TOO MANY DECIMAL PLACES

      TOO MANY INTEGER PLACES

b) The following messages are displayed without the picture string.

ILLEGAL KANJI DATA

NOT FLOATING STRING

OVERFLOW IN FLOATING DATA

UNDERFLOW IN FLOATING DATA

ILLEGAL ALPHABETIC DATA

c) After displaying one of the following messages, the cursor is repositioned at the beginning of the field in error.

ILLEGAL CHARACTER IN STRING

MORE THAN ONE DECIMAL POINT

MORE THAN ONE SIGN CHARACTER

SIGN NOT ALLOWED

SIGN NOT IN PROPER POSITION

TRIED TO STORE TOO MANY DIGITS

2. Peripheral device status messages resulting from I-O attempts on not ready devices.

DEVICE NOT READY

This message is issued during the execution of an OPEN statement for a floppy disk when it is not ready.

The user has two minutes to ready the device after which the status of the message changes to an error with an I-O status value of A2.

PRINTER NOT READY

This message is issued during the execution of an OPEN or a WRITE statement for a printer when it is in local mode or is turned off.

The user has two minutes to ready the device after which the status of the message changes to an error with an I-O status value of 96.

ERROR MESSAGES

The Run Time error messages consist of two or three lines as follows:

1. The first line of the message is:

   COBOL RUN TIME ERROR

2. For errors associated with INPUT-OUTPUT statements the second line is:

   (statement) * FILE IS (filename)

   Where:
   statement is one of the following: READ 1 (sequential access), READ 2 (random access), WRITE, START, REWRITE, DELETE, LOCK, UNLOCK, OPEN, CLOSE, RELEASE, RETURN, SORT, MERGE; filename is the filename contained in the ASSIGN TO phrase in the SELECT clause.

   For CALL and CANCEL statement errors, the second line is:

   CALL "STATEMENT", or

   CANCEL "STATEMENT"

   For SORT statement, the second line is as follows:

   SORT "STATEMENT"

3. If no USE procedure is specified for the file in which the I-O error occurred a third line is displayed with the I-O status value associated with the error in parentheses.

In the following list the messages are divided into those with a possible I-O status and those without.

COBOL RUN TIME ERROR
"statement" * FILE IS "filename"
PERMANENT ERROR: NO FURTHER INFORMATION AVAILABLE (30)

A permanent error exists and no further information is available concerning the input-output operation. (All Organizations).

COBOL RUN TIME ERROR
"statement" * FILE IS "filename"
REMOTE MACHINE NOT AVAILABLE (9B)

An I-O operation was attempted for a file resident on another machine, but the line is down. (All Organizations).

COBOL RUN TIME ERROR
"statement" * FILE IS "filename"
REMOTE MACHINE NOT AVAILABLE (A9)

An OPEN statement was attempted for a file resident on another machine but the line is down. (All Organizations).

COBOL RUN TIME ERROR
"statement" * FILE IS "filename"
SECURITY VIOLATION (A8)

An I-O operation was attempted on a file, but the access is not permitted. (All Organizations).

COBOL RUN TIME ERROR
CLOSE * FILE IS "filename"
LOGIC ERROR: FILE NOT OPEN (42)

An CLOSE was attempted for a file not in the open mode. (All Organizations).

COBOL RUN TIME ERROR
"statement" * FILE IS "filename"
INVALID REMOTE FILE CONNECTOR - MACHINE UP (AA)

Execution of an I-O instruction on a file residing on another computer has not been carried out because of an incorrect file connection. The line or computer then not available is restored but the file connection attempted before failure of the line or computer is no longer recognized. (All Organizations).

COBOL RUN TIME ERROR
DELETE * FILE IS "filename"
ATTEMPT TO ACCESS A RECORD THAT IS LOCKED (92)

A DELETE was attempted on a record which is locked. (Relative and Indexed Organizations)

COBOL RUN TIME ERROR
DELETE * FILE IS "filename"
DELETE FILE NOT LEGAL FOR DEVICES (95)

A DELETE FILE was attempted on a file which is not removable, i.e. a printer file. (Sequential Organization).

COBOL RUN TIME ERROR
DELETE * FILE IS "filename"
FILE IS NOT CLOSED: CAN NOT BE REMOVED (90)

A DELETE FILE was attempted on a file which is not closed, logic error. (All Organizations).

COBOL RUN TIME ERROR
DELETE * FILE IS "filename"
INVALID PATH-NAME (A0)

Invalid path-name detected during the execution of a DELETE FILE. (All Organizations).

COBOL RUN TIME ERROR
DELETE * FILE IS "filename"
LOGIC ERROR: NOT PRECEDED BY SUCCESSFUL READ (43)

In the sequential access mode, the last input-output statement executed for the associated file prior to the attempted DELETE was not a successfully executed READ. (All Organizations).

```
COBOL RUN TIME ERROR
DELETE  *  FILE IS  "filename"
LOGIC ERROR: USE PROCEDURE NOT SPECIFIED (49)
```

A DELETE was attempted on a file not open in the I-O mode.

```
COBOL RUN TIME ERROR
DELETE  *  FILE IS  "filename"
NAME NOT FOUND (98)
```

A DELETE FILE was attempted on a file which does not exist. (All Organizations).

```
COBOL RUN TIME ERROR
DELETE FILE *  FILE IS  "filename"
ILLEGAL PATH-NAME (A6)
```

Illegal path name for a DELETE FILE statement.
The path name contained in the variable specified in the ASSIGN clause does not comply with the syntax rules for path names. (All organizations).

```
COBOL RUN TIME ERROR
LOCK  *  FILE IS  "filename"
ATTEMPT TO LOCK A RECORD THAT IS ALREADY LOCKED (94)
```

Attempt to LOCK a record which is already locked. (Relative and Indexed Organizations)

```
COBOL RUN TIME ERROR
LOCK  *  FILE IS  "filename"
ATTEMPT TO LOCK/UNLOCK A RECORD WHICH DOES NOT EXIST (99)
```

A LOCK or UNLOCK was attempted, using an alternate key, on a record which does not exist. (Indexed Organization)

```
COBOL RUN TIME ERROR
LOCK  * FILE IS "filename"
OPTIONAL INPUT FILE NOT PRESENT (9C)
```

Execution of a LOCK instruction on a non-existent optional file has been attempted. (For Relative or Indexed Organization).

```
COBOL RUN TIME ERROR
LOCK  *  FILE IS  "filename"
FILE IS NOT OPEN (93)
```

A LOCK or UNLOCK was attempted on a file which is not open, logic error. (Relative and Indexed Organizations)

```
COBOL RUN TIME ERROR
LOCK,  *  FILE IS  "filename"
TOO MANY RECORDS LOCKED (97)
```

An attempt was made to exceed the maximum number of records permitted to be locked at one time. (Relative and Index Organizations)

```
COBOL RUN TIME ERROR
OPEN  *  FILE IS  "filename"
DEVICE NOT READY (A2)
```

An OPEN was attempted for a file on a device which is not ready.
Before this condition occurs, a warning message is displayed and the device is unavailable for a period of two minutes. (All Organizations)

```
COBOL RUN TIME ERROR
OPEN  *  FILE IS  "filename"
ATTEMPT TO LOCK A FILE ALREADY OPENED OR LOCKED (A1)
```

An OPEN was attempted on a file which is locked or an OPEN with LOCK was attempted on a file which is already opened by another process. (Relative and Indexed Organizations).

```
COBOL RUN TIME ERROR
OPEN  *  FILE IS  "filename"
INVALID PATH-NAME (A0)
```

Invalid path-name detected during the execution of an OPEN file. (All Organizations).

```
COBOL RUN TIME ERROR
OPEN  *  FILE IS  "filename"
LOGIC ERROR: FILE ALREADY OPEN (41)
```

An OPEN was attempted for a file in the open mode. (All Organizations)

COBOL RUN TIME ERROR
OPEN * FILE IS "filename"
OUT OF DISK SPACE (A7)

The execution of an OPEN statement caused the creation of a file when there is no space available on the medium (disk).
The user may change to a different medium or may execute DELETE FILE statements to create space.
(All Organization)

COBOL RUN TIME ERROR
OPEN * FILE IS "filename"
PERMANENT ERROR: NON-OPTIONAL FILE NOT PRESENT (35)

An OPEN with the INPUT, I-O, or EXTEND phrase was attempted on a non-optional file that is not present. (All Organizations).

COBOL RUN TIME ERROR
OPEN * FILE IS "filename"
PERMANENT ERROR: MUST BE MASS STORAGE FILE (37)

An OPEN statement was attempted on a file that is required to be a mass storage file but is not a mass storage file. (All Organizations).

COBOL RUN TIME ERROR
OPEN * FILE IS "filename"
PERMANENT ERROR: FILE CLOSED WITH LOCK (38)

An OPEN was attempted on a file previously closed with lock. (All Organizations).

COBOL RUN TIME ERROR
OPEN * FILE IS "filename"
PERMANENT ERROR: FILE NOT CONSISTENT WITH DECLARATION (39)

The OPEN was unsuccessful because a conflict has been detected between the fixed file attributes and the attributes specified for that file in the program. (All Organizations).

COBOL RUN TIME ERROR
OPEN * FILE IS "filename"
PRINTER CAN BE OPENED ONLY FOR OUTPUT (A3)

An OPEN with the INPUT phrase was attempted for a device which may not be open for input. (Sequential Organization).

COBOL RUN TIME ERROR
OPEN * FILE IS "filename"
PRINTER NOT READY (A2)

An OPEN was attempted for a file on a device which is not ready.
Before this condition occurs, a warning message is displayed and the device is unavailable for a period of two minutes. (All Organizations).

COBOL RUN TIME ERROR
OPEN * FILE IS "filename"
TEXT FILE CAN NOT BE OPENED IN I-O MODE (A4)

An OPEN with the I-O phrase was attempted for a text-file. (Sequential Organization).

COBOL RUN TIME ERROR
OPEN * FILE IS "filename"
TOO MANY FILES OPEN (A5)

An OPEN was attempted which would result in the maximum permitted number of files open at one time being exceeded. (All Organizations).

COBOL RUN TIME ERROR
OPEN * FILE IS "filename"
ILLEGAL PATH-NAME (A6)

Illegal path name for an OPEN statement.
The path name contained in the variable specified in the ASSIGN clause does not comply with the syntax rules for path names.
(All organizations).

```
COBOL RUN TIME ERROR
READ 1  *  FILE IS  "filename"
AT END: NO NEXT (PREVIOUS) RECORD EXISTS (10)
```

A sequential READ was attempted and no next (previous) logical record exists in the file because the end (beginning) of the file has been reached. (All Organizations).

```
COBOL RUN TIME ERROR
READ 1  *  FILE IS  "filename"
AT END: RELATIVE RECORD NUMBER OUT OF RANGE (14)
```

A sequential READ was attempted for a relative file and the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file. (Relative Organizations).

```
COBOL RUN TIME ERROR
READ 1  *  FILE IS  "filename"
AT END: OPTIONAL INPUT FILE NOT PRESENT (15)
```

A sequential READ was attempted for the first time on an optional input file that is not present. (All Organizations).

```
COBOL RUN TIME ERROR
READ 1  *  FILE IS  "filename"
AT END: AT END CONDITION ALREADY EXISTS (16)
```

A sequential READ was attempted and the at end condition already exists. (All Organizations).

```
COBOL RUN TIME ERROR
READ 1  *  FILE IS  "filename"
LOGIC ERROR: NOT PRECEDED BY SUCCESSFUL READ OR START (46)
```

A sequential READ was attempted on a file open in the INPUT or I-O mode and no valid next record has been established because:

- The preceding START was unsuccessful (Relative and Indexed Organization), or
- The preceding READ was unsuccessful but did not cause an at end condition. (All Organizations).

```
COBOL RUN TIME ERROR
READ 1  *  FILE IS  "filename"
PERMANENT ERROR: BOUNDARY VIOLATION (34)
```

Attempt to write beyond the externally defined boundaries of a sequential file (out of space for a magnetic tape). (Sequential Organization).

```
COBOL RUN TIME ERROR
READ 1  *  FILE IS  "filename"
PRINTER NOT READY (96)
```

A READ was attempted for a device not ready, system and work station printers only.
Before this condition occurs, a warning message was displayed and the device continued to be unavailable for a period of two minutes. (Sequential Organization).

```
COBOL RUN TIME ERROR
READ 2  *  FILE IS  "filename"
INV. KEY: KEY DOES NOT EXIST (23)
```

An attempt was made to randomly access a record that does not exist in the file. (Relative and Indexed Organizations).

```
COBOL RUN TIME ERROR
READ 2  *  FILE IS  "filename"
INV. KEY: OPTIONAL INPUT FILE NOT PRESENT (25)
```

A READ was attempted on an optional file that is not present. (Relative and Indexed Organizations).

```
COBOL RUN TIME ERROR
READ 2  *  FILE IS  "filename"
LOGIC ERROR: FILE NOT OPEN IN INPUT OR I-O (47)
```

A READ was attempted on a file not open in the INPUT or I-O mode. (All Organizations).

```
COBOL RUN TIME ERROR
REWRITE  *  FILE IS  "filename"
ATTEMPT TO ACCESS A RECORD THAT IS LOCKED (92)
```

A REWRITE was attempted on a record which is locked. (Relative and Indexed Organizations).

```
COBOL RUN TIME ERROR
REWRITE  *  FILE IS  "filename"
INV. KEY: PRIMARY KEY VALUES OUT OF SEQUENCE (21)
```

A sequence error exists for a sequentially accessed indexed file. The prime record key value has been changed by the program between the successful execution of a READ and the execution of the REWRITE for the file, or the ascending sequence requirements for successive record key values are violated. (Indexed Organization).

```
COBOL RUN TIME ERROR
REWRITE  *  FILE IS  "filename"
INV. KEY: KEY ALREADY EXISTS (22)
```

An attempt was made to rewrite a record that would create a duplicate alternate record key without the DUPLICATES phrase specified. (Indexed Organization).

```
COBOL RUN TIME ERROR
REWRITE  *  FILE IS  "filename"
LOGIC ERROR: USE PROCEDURE NOT SPECIFIED (43)
```

In the sequential access mode, the last input-output statement executed for the associated file prior to the attempted REWRITE was not a successfully executed READ. (All Organizations).

```
COBOL RUN TIME ERROR
REWRITE  *  FILE IS  "filename"
LOGIC ERROR: FILE NOT OPEN IN I-O (49)
```

A REWRITE was attempted on a file not open in the I-O mode.

```
COBOL RUN TIME ERROR
START  *  FILE IS  "filename"
INV. KEY: OPTIONAL INPUT FILE NOT PRESENT (25)
```

A random START was attempted on an optional file that is not present. (Relative and Indexed Organizations).

```
COBOL RUN TIME ERROR
START  *  FILE IS  "filename"
LOGIC ERROR: FILE NOT OPEN IN INPUT OR I-O (47)
```

A START was attempted on a file not open in the INPUT or I-O mode. (All Organizations).

COBOL RUN TIME ERROR
UNLOCK * FILE IS "filename"
OPTIONAL INPUT FILE NOT PRESENT (9C)

Execution of an UNLOCK instruction on a non-existent optional file has been attempted. (For Relative or Indexed organizations).

COBOL RUN TIME ERROR
WRITE * FILE IS "filename"
ATTEMPT TO ACCESS A RECORD THAT IS LOCKED (92)

A WRITE was attempted on a record which is locked. (Relative and Indexed Organizations).

COBOL RUN TIME ERROR
WRITE * FILE IS "filename"
INV. KEY: KEY ALREADY EXISTS (22)

An attempt was made to write a record that would create a duplicate key (relative), or duplicate prime record key, or to write a record that would create a duplicate alternate record key without the DUPLICATES phrase specified. (Indexed Organization).

COBOL RUN TIME ERROR
WRITE * FILE IS "filename"
INV. KEY: OUT OF DISK SPACE OR RECORD NUMBER OUT OF RANGE (24)

An attempt was made to write beyond the externally defined boundaries of an indexed or relative file (out of space on the medium). Or, a sequential WRITE was attempted for a relative file and the number of significant digits in the relative record number is larger than the size of the relative key data item described for the file. (Relative and Indexed Organizations).
In the first case the user may execute DELETE FILE statements to make space available. Space must be available before any WRITE statement for the file can be successfully executed.

COBOL RUN TIME ERROR
WRITE * FILE IS "filename"
LOGIC ERROR: FILE NOT OPEN IN I-O, OUTPUT OR EXTENDED (48)

A WRITE was attempted on a file not open in the I-O, OUTPUT or EXTEND mode. (All Organizations).

```
COBOL RUN TIME ERROR
WRITE  *  FILE IS  "filename"
OPEN MODE I-O ILLEGAL FOR SEQUENTIAL ACCESS FILE (91)
```

A WRITE was attempted for a file open in the I-O mode with sequential access. (Relative and Indexed Organizations).

```
COBOL RUN TIME ERROR
WRITE  *  FILE IS  "filename"
OUT OF DISK SPACE (9A)
```

A WRITE was attempted on a file which is complete and there is no more disk space available. (Sequential Organization)
The user may execute DELETE FILE statements to create space. The message is displayed until the space becames available.

```
COBOL RUN TIME ERROR
WRITE  *  FILE IS  "filename"
PERMANENT ERROR: BOUNDARY VIOLATION (34)
```

Attempt to write beyond the externally defined boundaries of a sequential file (out of space for a magnetic tape). (Sequential Organization).

```
COBOL RUN TIME ERROR
WRITE  *  FILE IS  "filename"
PRINTER NOT READY (96)
```

A WRITE was attempted for a device not ready, system and work station printers only.
Before this condition occurs, a warning message was displayed and the device continued to be unavailable for a period of two minutes. (Sequential Organization).

INDEX

4. INTERFACCE COBOL CON ALTRI LINGUAGGI E SERVIZI

SERVIZI

I programmi scritti in COBOL possono essere linkati anche con programmi scritti in altri linguaggi o con altri servizi. L'accesso ad essi avviene sempre eseguendo un'istruzione CALL nel programma e facendo il link del codice oggetto con la libreria di interfaccia che contiene i riferimenti ai servizi.

NOTA: Quando si interfacciano altri servizi, i tasti funzione assumono il significato specifico di quei servizi. Per tali significati si consultino i manuali relativi ai singoli servizi.

COBOL E SERVIZI MTS

La struttura dei dati nella frase USING nell'istruzione CALL deve essere definita in accordo con ciascuno dei servizi richiesti. Lo speciale registro, MESSAGE, viene aggiornato con un codice che indica l'avvenuta esecuzione dell'istruzione CALL. Per una completa descrizione dei servizi consultare il manuale MTS Guida del Programmatore.

I programmi applicativi COBOL e i servizi scritti dall'utente devono essere compilati per l'ambiente transazionale, specificando l'opzione 'TH' per il compilatore COBOL.

I servizi standard di MTS sono ottenuti facendo il link del programma con le interfacce MTS_ui.obj, MTS.obj, MTS_BATCH.obj, CHAIN_DML.obj e MWS_ui.obj. Queste sono incluse nella libreria EXTINTF.LIB che viene linkata ai programmi COBOL degli utenti usando i comandi del linker presenti nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

I servizi scritti dall'utente devono essere linkati individualmente, formando ognuno un modulo di programma caricabile individualmente.

Prima dell'esecuzione del programma applicativo, deve essere abilitato l'accesso ai servizi scritti dall'utente. Durante la configurazione dell'ambiente per MTS deve essere caricato il componente 'TU MANAGER' ed i nomi, i path name e le caratteristiche dei servizi scritti dall'utente devono essere introdotti in una tabella mediante l'utility 'MTSCONF'. Questa tabella viene usata dalle routine di run time per risolvere le chiamate ai servizi. Per ulteriori informazioni consultare il manuale MTS Procedure di Configurazione.

COBOL E SERVIZI VISA COMPATIBILI S6000

I dati nella frase USING dell'istruzione CALL devono essere definiti in accordo con i servizi richiesti; per gli esempi ed una descrizione completa consultare il manuale VISA Versione S6000 Guida del Programmatore.

Il file DDPARM deve esistere nella directory di lavoro corrente e deve contenere il path name completo delle librerie dei formati, primo carattere '/'.

Non sono richieste opzioni di compilazione.

Non sono richieste opzioni per il linker. I servizi del VISA S6000 sono inclusi tra le routine di run time dell'ambiente COBOL, se questo li prevede. I comandi per linkare i servizi del VISA S6000 ai programmi COBOL si trovano nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL E SERVIZI VISA

I dati nella frase USING dell'istruzione CALL devono essere definiti in accordo con i servizi richiesti; per esempi ed una descrizione completa, consultare il manuale VISA - Package per le Gestioni dei Formati - Guida del Programmatore.

Non sono richieste opzioni di compilazione.

I servizi VISA possono essere ottenuti facendo il link del programma con l'interfaccia VISA.obj che fa parte della libreria EXTINTF.LIB e che puo' essere linkata ai programmi utente per mezzo dei comandi del linker che si trovano nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

I servizi VISA MULTIPROCESS possono essere ottenuti facendo il link del programma con l'interfaccia VISAP.obj che fa parte della libreria EXTINTF.LIB che puo'essere linkata ai programmi utente per mezzo dei comandi del linker che si trovano nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL E I SERVIZI LINE MONITOR

I dati nella frase USING dell'istruzione CALL devono essere definiti in accordo con i servizi richiesti; per esempi ed una descrizione completa consultare il manuale CSS - Gestione della Linea - Guida del Programmatore.

Non sono richieste opzioni di compilazione.

I servizi LINE MONITOR possono essere ottenuti facendo il link del programma con l'interfaccia LM_ui.obj che fa parte della libreria EXTINTF.LIB e che puo' essere linkata ai programmi utente per mezzo dei comandi del linker che si trovano nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL E I SERVIZI BEAM

I dati nella frase USING dell'istruzione CALL devono essere definiti in accordo con i servizi richiesti; per esempi ed una descrizione completa consultare il manuale BEAM - Business Environment Application Monitor - Guida dell'Utente.

Non sono richieste opzioni di compilazione.

I servizi BEAM possono essere ottenuti facendo il link del programma con le interfacce bconfl.obj e blog.obj che fanno parte della libreria EXTINTF.LIB e che puo' essere linkata ai programmi utente per mezzo dei comandi del linker che si trovano nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL E IL PGU - PACKAGE PER LA GESTIONE DELLA GRAFICA

I dati nella frase USING dell'istruzione CALL devono essere definiti in accordo con i servizi richiesti; per esempi ed una descrizione completa consultare il manuale PGU - Package per la Gestione della Grafica - Guida del Programmatore.

Non sono richieste opzioni di compilazione.

I servizi grafici del PGU possono essere ottenuti facendo il link del programma col codice oggetto relativo alla parte grafica e i file delle tabelle relative al tipo di caratteri; questi ultimi si chiamano FONT0...FONT8 e fanno parte della directory /IPL/DPC/GRAPHICS. Nel link deve anche essere inclusa l'interfaccia CO_PGU_ui.obj che risiede nella libreria EXTINTF.LIB, che viene linkata ai programmi COBOL degli utenti usando i comandi del linker presenti nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

COSTRUZIONE DI GRIGLIE SULLO SCHERMO

E' possibile da un programma COBOL inviare sullo schermo una griglia mediante utilizzo dei caratteri della semigrafica semplice.

I dati nella frase USING dell'istruzione CALL devono essere definiti in accordo con i servizi richiesti; per esempi ed una descrizione completa consultare il manuale MOS - Guida del Programmatore.

Non sono richieste opzioni di compilazione e non e' necessario includere alcuna interfaccia.

COBOL E SERVIZI DI RICOVERO

I servizi di ricovero vengono realizzati mediante il Commit Manager che fornisce strumenti per mantenere la coerenza in una Base di Dati a fronte di una sequenza di operazioni (transazione).

Le procedure di ricovero sono richieste mediante l'istruzione CALL con il nome del servizio ed i dati introdotti dalla frase USING devono essere in accordo con il servizio stesso. Per esempi ed una completa descrizione dei servizi, consultare il manuale MOS - Guida del Programmatore.

Non sono richieste opzioni di compilazione.

I servizi standard del Commit Manager sono ottenuti facendo il link del programma con l'interfaccia CM_ui.obj che risiede nella libreria EXTINTF.LIB, che viene linkata ai programmi COBOL degli utenti usando i comandi del linker presenti nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL E VERIFICA DELLE FIRME

Le procedure di verifica firme consentono l'acquisizione e la visualizzazione di una firma.

I dati della frase USING dell'istruzione CALL devono essere definiti in accordo con i servizi richiesti; per esempi ed una descrizione completa consultare il manuale MOS - Guida del Programmatore.

Non sono richieste opzioni di compilazione.

I servizi di verifica delle firme possono essere ottenuti eseguendo il link del programma con le interfacce tubo.obj, visu.obj, decomp.obj e clear.obj che risiedono della libreria EXTINTF.LIB, che puo' essere linkata ai programmi COBOL degli utenti usando i comandi del linker presenti nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL E SERVIZI DI MESSAGE SWITCHING

I servizi di Message Switching permettono di scambiare messaggi fra applicativi attivi residenti su nodi diversi di una configurazione distribuita.

I dati della frase USING dell'istruzione CAll devono essere definiti in accordo con i servizi richiesti; per esempi e una descrizione completa consultare il manuale MOS - Guida del Programmatore.

Non sono richieste opzioni di compilazione.

I servizi di Message Switching sono ottenuti eseguendo il link del programma con l'interfaccia MSW_ui.obj che risiede nella libreria EXTINTF.LIB, che puo' essere concatenata ai programmi COBOL usando i comandi del linker presenti nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL E SERVIZI DI ERROR LOGGING

La prestazione di ERROR LOGGING e' fornita da LMS (Local Management System), i cui moduli devono essere presenti nella configurazione del sistema per poter usufruire di questa prestazione.

I dati introdotti dalla frase USING dell'istruzione CALL devono essere in accordo con il servizio stesso. Per esempi ed una completa descrizione dei servizi, consultare il manuale MOS - Guida del Programmatore.

Non sono richieste opzioni di compilazione.

I servizi standard di LMS sono ottenuti facendo il link del programma con l'interfaccia lms_ui.obj che risiede nella libreria EXTINTF.LIB, che viene linkata ai programmi COBOL degli utenti usando i comandi del linker presenti nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL E COLLEGAMENTO IN RETE ONE

Per usufruire dei servizi offerti dal collegamento in rete ONE occorre accedere o al modulo SLAM o al modulo SNAM mediante l'istruzione CALL con il nome della direttiva richiesta ed i dati nella frase USING devono essere in accordo con il servizio stesso.

Per esempi ed una completa descrizione dei servizi, consultare il manuale MOS - ONE Guida dell'Utente.

Non sono richieste opzioni di compilazione.

I servizi standard del modulo SLAM sono ottenuti eseguendo il link del programma utilizzando il file /IPL/DPC/COB_RTS/LIB/RTLINK.

I servizi standard del modulo SNAM sono ottenuti eseguendo il link del programma utilizzando il file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL E COLLEGAMENTO AL CAT

E' possibile da un programma applicativo COBOL eseguire un collegamento con il posto di lavoro, usato in applicazioni bancarie, CAT (Customer Activated Terminal). Per la descrizione di quest'ultimo e per le modalita' programmative da seguire, consultare il manuale CAT - Interfaccia Applicativa - Guida del Programmatore.

Non sono richieste opzioni di compilazione.

I servizi standard del CAT sono ottenuti facendo il link del programma con le interfacce eru_ui.obj, INTAPP.obj e INTAMB.obj che risiedono nella libreria EXTINTF.LIB, che viene linkata ai programmi COBOL degli utenti usando i comandi del linker presenti nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL E I SERVIZI DI LOGOFF

L a funzione LOGOFF permette le sospensioni del collegamento, su linea switched fra un posto di lavoro remoto e il sistema, consentendo un risparmio sulla tariffa telefonica.

I dati della frase USING dell'istruzione CALL devono essere definiti in accordo con i servizi richiesti; per esempi ed una descrizione completa consultare il manuale MOS - Guida del Programmatore.

Non sono richieste opzioni di compilazione.

I servizi di LOGOFF sono ottenuti eseguendo il link del programma con l'interfaccia WRS.lib che risiede nella libreria EXTINTF.LIB che puo' essere concatenata ai programmi COBOL degli utenti usando i comandi del linker specificati nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL E RS232/CL

Permette di gestire periferiche collegate all'interfaccia RS232/CL.

I dati della frase USING dell'istruzione CALL devono essere definiti in accordo con i servizi richiesti; per esempi ed una descrizione completa consultare il manuale Interfaccia RS232/CL - Guida del Programmatore.

Non sono richieste opzioni di compilazione.

I servizi di RS232/CL sono ottenuti eseguendo il link del programma con le interfaccie TUB_ui.obj e rs_ui.obj che risiedono nella libreria EXTINTF.LIB, che puo' essere concatenata ai programmi COBOL degli utenti usando i comandi del linker presenti nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

COBOL E IL GSP - GRAPHING SUBROUTINE PACKAGE

Il package GSP realizza output grafici orientati ad applicazioni commerciali.

I dati della frase USING dell'istruzione CALL devono essere definiti in accordo con i servizi richiesti; per esempi e una descrizione completa consultare il manuale GSP - GRAPHING SUBROUTINE PACKAGE - Application Interface Programmer Guide.

Non sono richieste opzioni di compilazione.

I servizi di GSP sono ottenuti eseguendo il link del programma con l'interfaccia RTGSP_ui.obj che risiede nella libreria EXTINTF.LIB, che puo' essere concatenata ai programmi COBOL degli utenti usando i comandi del linker presenti nel file /IPL/DPC/COB_RTS/LIB/RTLINK.

DA COBOL A PASCAL+

La natura e la filosofia dei due linguaggi e' totalmente diversa.

Nel PASCAL+ c'e' il concetto di programma e di procedura/funzione esportati dal modulo. Nel COBOL non c'e' differenziazione. Nel senso del PASCAL ogni programma COBOL puo' essere sia un 'programma' che una 'procedura'; non c'e' modo di definire una funzione.

In PASCAL+ c'e' il concetto di chiamata per valore e il concetto di chiamata per riferimento.

I parametri, che mediante una frase USING devono essere passati ad una procedura PASCAL+, debbono essere mappati con cura, tenendo conto del tipo e della loro rappresentazione interna che e' nota all'utente, in COBOL, ma non in PASCAL+.

Tutti i parametri coinvolti nelle chiamate, dal punto di vista del PASCAL+ devono essere "parametri passati per riferimento", allo scopo di poter fare riferimento correttamente ai parametri all'interno del programma e della procedura chiamata.

Non sono richieste opzioni per la compilazione COBOL.

Non sono richieste opzioni per i passi del link, ma i due file di codice oggetto devono essere linkati assieme nello stesso modulo di caricamento.

ENTITA' RICHIAMABILI E CONVENZIONI PER I NOMI

Da un programma COBOL e' possibile chiamare:

1. Una procedura PASCAL+ estratta da un modulo. In questo caso, il literal o il nome di dato nell'istruzione CALL deve contenere:

   identificatore del modulo - identificatore di procedura

   come in

   ```
   CALL "MODULEY_PROCEDUREX" USING PARAMETER-1 PARAMETER-2 ...
   ```

2. Una procedura PASCAL+ a livello globale in un modulo compilato usando l'opzione {$e+}. Il literal o il nome di dato nell'istruzione CALL deve contenere:

   identificatore di procedura

   come in

   ```
   CALL "PROCEDUREX" USING PARAMETER-1 PARAMETER-2 ...
   ```

CORRISPONDENZA TRA I PARAMETRI

Tutti i parametri coinvolti nelle CALL, dal punto di vista del Pascal+, devono essere "parametri passati per riferimento". Per fare correttamente riferimento ai parametri interni alla procedura chiamata, si devono seguire alcune regole addizionali.

Se all'interno della procedura o del programma non si fa riferimento ai parametri, si possono usare, per il PASCAL+ il tipo carattere (char) mentre per il COBOL dati definiti con PIC X USAGE DISPLAY.

CORRISPONDENZA TRA I DATI ELEMENTARI DEL COBOL E I TIPI SEMPLICI DEL PASCAL+

La tabella specifica tutte le possibili sostituzioni e le regole ad esse associate, che devono essere seguite nella dichiarazione dei parametri di un programma COBOL. Nella tabella non sono presenti i tipi di dati del COBOL che non hanno alcuna corrispondenza nei tipi del PASCAL+. Nella tabella, la presenza del simbolo 'N' sta ad indicare una combinazione non permessa; al simbolo 'Y' che indica le combinazioni permesse, viene associato un numero che indica la regola a cui deve sottostare la combinazione.

tipi del PASCAL+ * COBOL * tipo e uso	Alfanumerici	Alfabetici DISPLAY	Numerici	Numerici COMP	Numerici COMP-4	Numerici COMP-2
char	Y 1	Y 1	Y 1	N	N	N
integer	N	N	N	Y 2	Y 2	N
longinteger	N	N	N	Y 3	Y 3	N
boolean	N	N	N	Y 4	N	N
enumerated types	N	N	N	Y 5	N	N
integer-subrange	N	N	N	Y 6	N	N
real	N	N	N	N	N	Y 7

Dati Elementari del COBOL - Sostituzione con i Tipi Semplici del PASCAL+

Regole per la Descrizione dei Dati del COBOL

1. Una PICTURE deve contenere almeno un carattere; ad esempio:

```
01 NAME-1 PIC X [USAGE DISPLAY].
01 NAME-2 PIC A [USAGE DISPLAY].
01 NAME-3 PIC 9 [USAGE DISPLAY].
```

dove: PIC X puo' mappare qualsiasi dato di tipo char del PASCAL+; PIC A puo' mappare qualsiasi dato di tipo alfabetico; PIC 9 e' in grado di mappare tutti i caratteri numerici o cifre.

2. La descrizione dei dati deve essere:

```
01 NAME-1 PIC S9(4) [USAGE IS] COMP.  o
01 NAME-2 PIC S9(4) [USAGE IS] COMP-4
```

Nota: L'insieme dei valori disponibili va da -9999 a +9999.

3. La descrizione dei dati deve essere:

```
01 NAME-1 PIC S9(9) [USAGE IS] COMP.  o
01 NAME-2 PIC S9(9) [USAGE IS] COMP-4.
```

Nota: L'insieme dei valori disponibili va da -999999999 a +999999999.

4. La descrizione dei dati deve essere:

```
01 NAME-1 PIC 9 [USAGE IS] COMP.
```

I valori associati sono i numeri ordinali del PASCAL+, "vero" o "falso".

5. Per i tipi enumerati (enumerated types) che includono almeno 99 elementi, la descrizione dei dati deve essere:

```
01 NAME-1 PIC 9(2) [USAGE IS] COMP.
```

Il valore associato ad un elemento e' il numero d'ordine di quell'elemento.

6. Per i sottoinsiemi -LIM..LIM dove LIM e' minore o uguale a 99, la descrizione dei dati deve essere:

```
01 NAME-1 PIC S9(2) [USAGE IS] COMP.
```

7. La descrizione dei dati deve essere:

```
01 NAME-1 [USAGE IS] COMP-2.
```

CORRISPONDENZA TRA I DATI (ELEMENTARI O GRUPPI) DEL COBOL E QUELLI DI TIPO ARRAY DEL PASCAL+

La tabella mostra tutte le possibili sostituzioni dei parametri di tipo array. E' importante notare che non c'e' alcuna corrispondenza o adattabilita' dei dati di tipo array del PASCAL+ e le descrizioni dei dati del COBOL.

tipi del PASCAL+ * COBOL * tipo e uso	Alfanumerici	Alfabetici DISPLAY	Numerici	Numerici COMP	Numerici COMP-4	Numerici COMP-2
packed or unpacked array of char	Y 1	Y 1	Y 1	N	N	N
array of integer	N	N	N	Y 2	Y 2	N
array of longinteger	N	N	N	Y 3	Y 3	N
packed array of types enumerated	N	N	N	Y 4	N	N
packed array of integer subrange	N	N	N	Y 5	N	N
array of real	N	N	N	N	N	Y 6

Equivalenza tra i dati (elementari o gruppi) del COBOL e i Dati di Tipo Array del PASCAL+

Regole per la Descrizione dei Dati del COBOL

1. In COBOL la descrizione di una stringa o di un registro e' ancora un dato elementare, con una PICTURE che contiene tanti caratteri quanti sono gli elementi del Pascal+ di tipo packed array. Ad esempio:

   01 BUFFER PIC X(240) [USAGE DISPLAY].

   che equivale alla variabile Pascal+:

   type buffer = array[1..240] of char;

   Nota: Per i dati numerici di tipo DISPLAY, il numero massimo di "9" che una PICTURE puo' contenere e' 18, mentre il numero massimo di "X" o di "A" che possono essere contenuti rispettivamente in una PICTURE alfabetica o alfanumerica e' 32766.

   Per le tabelle del COBOL, la descrizione e' un gruppo contenente un elemento descritto mediante la clausola OCCURS ed il cui numero di occorrenze deve essere uguale al numero degli elementi della matrice in PASCAL+.

Ad esempio

```
01 PARAMETER-1.
   02 NAME-1 OCCURS 240 TIMES PIC X [USAGE DISPLAY].
```

che equivale all'elemento Pascal+ descritto nell'esempio precedente.

Nota: In COBOL, i valori subscritti cominciano sempre per 1, mentre in Pascal cominciano sempre con un numero ordinale.

2. La descrizione dei dati del COBOL deve essere un gruppo che contiene soltanto un elemento descritto come indicato nella regola 2 descritta in DATI ELEMENTARI DEL COBOL mediante la clausola OCCURS. Ad esempio, una variabile Pascal+ di tipo:

```
type fourIntegers = array[0..3] of integer;
```

equivale alla descrizione COBOL

```
01 PARAMETER-1.
   02 TABLE-OF-INTEGERS OCCURS 4 TIMES PIC S9(4) USAGE COMP-4.
```

3. Per gli array di tipo longinteger si applica la regola descritta al punto due, sostituendo la PICTURE con:

```
PIC S9(9)
```

4. La descrizione del dato deve essere un gruppo contenente una tabella di dati dichiarati come descritto al punto 5.

Il tipo Pascal+ :

```
type vowels = (a, e, i, o, u);
     vowel_table = packed array[1..16] of vowels;
```

in COBOL, e' equivalente alla descrizione:

```
01 NAME-1.
   02 VOWELS-TABLE PIC 9(2) COMP OCCURS 16 TIMES.
```

5. Ad un array of integer subranges si deve applicare quanto descritto al punto 4, sostituendo la PICTURE con:

```
PIC S9(2)
```

6. La descrizione del dato deve essere un gruppo contenente una tabella di dati dichiarati come descritto al punto 7. Ad esempio:

```
01 NAME-1.
   02 TABLE-OF-REALS OCCURS 10 TIMES
     [USAGE IS] COMP-2.
```

CORRISPONDENZA TRA I GRUPPI DI DATI DEL COBOL E IL TIPO RECORD DEL PASCAL+

Tra i record types del Pascal+, si possono prendere in considerazione solo quelli di tipo "unpacked". Non bisogna mappare i gruppi di dati del COBOL nei record di tipo packed del PASCAL+ poiche' l'algoritmo che realizza i file packed cambia da una implementazione ad un'altra.

Si raccomanda l'uso della mappa dei dati nell'ordine presentato dal compilatore.

Per mappare i record di tipo "unpacked", bisogna seguire le seguenti regole:

1. L'allocazione dello spazio di memoria viene eseguita nell'ordine in cui i dati vengono dichiarati sia dal compilatore COBOL che dal compilatore PASCAL+.

2. In PASCAL+ i simple types richiedono un numero pari di byte cosi' come integer e longinteger sono parole (word-aligned). Le descrizioni dei corrispondenti dati elementari del COBOL, che erano state indicate nella tabella DATI ELEMENTARI DEL COBOL, devono contenere la clausola SYNCHRONIZED.

3. I tipi semplici del Pascal+ che richiedono un byte, come i tipi, boolean, enumerated types, e integer subranges, vengono allocati nell'indirizzo piu' basso disponibile. In tale caso la clausola SYNCHRONIZED non deve essere presente nella descrizione dell'elemento corrispondente.

4. I record con varianti del Pascal+ , in COBOL, possono essere mappati nei gruppi di dati contenenti la clausola REDEFINES per ognuna delle parti variabili. Per ottenere cio' si devono raggruppare le parti variabili in un dato strutturato ad esempio un record, cosi' che le parti variabili possano essere allocate a partire sempre da un indirizzo pari.

ESEMPI

1. Il record Pascal+ :

```
type vowel = (a, e, i, o, u)

type examp1 = record
     int1: integer;
     long1: longinteger;
     char1: char;
     intsub1: -99..99;
     flag1: boolean;
     int2: integer;
     entype1: vowels;
     end;
```

e' mappato nella seguente descrizione di record del COBOL:

```
01 EXAMP1-GROUP.
   02 INT1 PIC S9(4) COMP-4 SYNC.
   02 LONG1 PIC S9(9) COMP-4 SYNC.
   02 CHAR1 PIC X.
   02 INTSUB1 PIC S9(2) COMP.
   02 FLAG1 PIC 9 COMP.
   02 INT2 PIC S9(4) COMP-4 SYNC.
   02 ENTYPE1 PIC 9(2) COMP.
```

2. Il record con varianti Pascal+ :

```
type variants =
     (long, twoInts, fourChars, fourFlags);

type T_four_Characters = record
     char1, char2, char3, char4: char;
     end;

type T_four_Flags = record
     flag1, flag2, flag3, flag4: boolean;
     end;

type examp3 = record
     commonPart: integer;
     case tag_field : variants of
     long:
          (long1: longinteger);
     twoInts:
          (int1, int2: integer);
     fourChars:
          (four_Characters: T_four_Characters);
     fourFlags:
          (four_Flags: T_four_Flags);
     end;
```

e' mappato nella seguente descrizione di record del COBOL:

```
01 EXAMP2-GROUP.
   03 COMMONPART PIC S9(4) COMP-4.
   03 TAG-FIELD PIC 9(2) COMP.
   03 LONG1 PIC S9(9) COMP-4 SYNC.
   03 TWOINTS REDEFINES LONG1.
      06 INT1 PIC S9(4) COMP-4 SYNC.
      06 INT2 PIC S9(4) COMP-4 SYNC.
   03 FOUR-CHARACTERS REDEFINES LONG1.
      06 CHAR1 PIC X.
      06 CHAR2 PIC X.
      06 CHAR3 PIC X.
      06 CHAR4 PIC X.
   03 FOUR-FLAGS REDEFINES LONG1.
      06 FLAG1 PIC 9 COMP.
      06 FLAG2 PIC 9 COMP.
      06 FLAG3 PIC 9 COMP.
      06 FLAG4 PIC 9 COMP.
```

mentre le descrizioni seguenti, apparentemente equivalenti:

```
type variants =
     (long, twoInts, fourChars, fourFlags);

type examp2 = record
     commonPart: integer;
     case tag_field : variants of
     long:
          (long1: longinteger);
     twoInts:
          (int1, int2: integer);
     fourChars:
          (char1, char2, char3, char4: char);
     fourFlags:
          (flag1, flag2, flag3, flag4: boolean);
     end;
```

in COBOL, non possono essere mappate nella stessa descrizione di record.

REGOLE DI GUIDA ALLA SCRITTURA DELLE PROCEDURE PASCAL+

Queste regole devono essere seguite nella stesura di routine, scritte in PASCAL+, che devono essere richiamate da programmi COBOL.

- Scrivere solo procedure e non funzioni che non possono essere richiamate.
- Non usare come parametri gli array di tipo "flexible" o "conformant".
- Non usare parametri passati per valore.
- Usare solo i tipi di parametri descritti nella tabella precedente.
- Non usare come parametri record di tipo packed poiche', se questo una volta puo' essere mappato, non e' sempre corretto.
- Evitare per quanto possibile l'uso di record con varianti come parametri, poiche' questo tipo di mappaggio talvolta causa problemi.

PACKAGE COME GRAFICA, O LINE MONITOR

Si puo' accedere direttamente alle routine di sistema del PASCAL+ poiche' queste sono disponibili nella libreria COB_RTS/LIB/EXTINTF/LIB.

DAL PASCAL+ AL COBOL

Inversamente, da una unita' di compilazione PASCAL+ e' possibile chiamare un programma COBOL al livello piu' esterno. Da un programma principale PASCAL+, identificato dalla parola chiave PROGRAM, non e' possibile richiamare routine COBOL.
Il solo requisito e' che una qualsiasi routine PASCAL+ venga inclusa e che durante il link sia presente una dichiarazione di procedura del programma COBOL che contenga la direttiva "EXTERNAL", consultare il capitolo COMUNICAZIONE TRA PROGRAMMI del manuale COBOL - Manuale di Consultazione .
Il PASCAL+ richiede che una qualsiasi routine usata venga inclusa durante il link dall'interfaccia e non dalla libreria PASCAL+; consultare il manuale PASCAL+ Preparazione ed Esecuzione Programmi .